

AD-A000 000

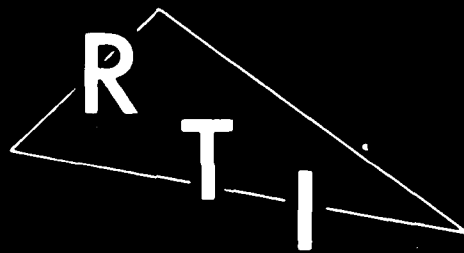
RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/G 12/1
TECHNIQUES FOR ON-LINE FAULT MONITORING IN MODULAR DIGITAL SYST--ETC(U)
JAN 80 J W GAULT, K S TRIVEDI, P N MARINOS N00039-78-C-0431
RTI/1667/00-01F NL

UNCLASSIFIED

1 of 1
00000000







RESEARCH TRIANGLE INSTITUTE

RTI/1667/00-01F

LEVEL

January 1980

ADA 084888

**TECHNIQUES FOR ON-LINE
FAULT MONITORING IN MODULAR
DIGITAL SYSTEMS**

Final Technical Report

Prepared for

Naval Electronic Systems Command
Code 304
Washington, D.C.

Under

Contract No. N00039-78-C-0431

Prepared by

Systems and Measurements Division
Research Triangle Institute
Research Triangle Park, NC 27709

**DTIC
ELECTE**
MAY 28 1980

**THIS DOCUMENT IS BEST QUALITY PRACTICE
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

"Approved for public release; distribution unlimited"

RESEARCH TRIANGLE PARK, NORTH CAROLINA 27709

DOC FILE COPY

DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DTIC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-A084 888	9
4. TITLE (and Subtitle)		5. TYPE OF REPORT & PERIOD COVERED
Techniques for On-Line Fault Monitoring in Modular Digital Systems.		Final Report. 15 August 1978 - 31 Dec. 1979
7. AUTHOR(s)		6. PERFORMING ORG. REPORT NUMBER
J. W. Gault K. S. Trivedi P. N. Marinos		RTI/1667/00-01F/
8. CONTRACT OR GRANT NUMBER(s)		
N00039-78-C-0431		
9. PERFORMING ORGANIZATION NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Research Triangle Institute Research Triangle Park, NC 27709		
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE
Naval Electronic Systems Command Code 304 Washington, DC 20360		January 1980
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES
		15. SECURITY CLASS. (of this report)
		Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Built-In-Test Self Repair Statistical Fault Monitoring Cellular Arrays		
Reliability Analysis Maintainability Analysis		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
<p>As digital electronic systems have become more complex, the importance of techniques for detecting faults using internal system resources has increased. Viable alternatives to heretofore ad hoc built-in-test approaches are described in this report. The following fault detection techniques and assessment methods are discussed:</p>		

80 5 23 011

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

- 1, Sample Fault Monitoring,
- 2, Programmable Cellular Arrays with Hardware Encoded Built-In-Test Facilities,
- (3) Mathematical Models for the Design and Analysis of On-Line Built-In-Testing,
- (4) A Flexible Fault-Tolerant Multiprocessor Architecture for Video Graphics.

Accession For	
NTIS	ORNL
DDC TAB	
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Dist	Available for special
A	77

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TECHNIQUES FOR ON-LINE
FAULT MONITORING IN MODULAR
DIGITAL SYSTEMS

Final Technical Report

Prepared for
Naval Electronics Systems Command
Code 304
Washington, D.C.

Under
Contract No. N00039-78-C-0431

Prepared by
Systems and Measurements Division
Research Triangle Institute
Research Triangle Park, NC 27709

January 1980

TABLE OF CONTENTS

	<u>Page</u>
LIST OF FIGURES	vii
LIST OF TABLES	ix
ACKNOWLEDGEMENTS	xi
<u>SECTION I - SAMPLE FAULT MONITORING (NCSU)</u>	I-1
1.0 INTRODUCTION	I-3
2.0 REFERENCELESS ON-LINE MONITORING USING OUTPUT STATISTICS . . .	I-5
2.1 Introduction	I-5
2.2 Fault Detection Strategy	I-6
2.3 Evaluating Effectiveness	I-8
2.4 Status and Conclusions	I-9
REFERENCES	I-10
3.0 SEQUENTIAL CIRCUIT OUTPUT PROBABILITIES FROM NETWORK REALIZATIONS	I-11
3.1 Introduction	I-11
3.2 Output Probabilities for Common Functions	I-12
3.3 Output Probabilities for General Composite Networks . . .	I-25
3.4 Summary	I-29
REFERENCES	I-30
4.0 A STATISTICAL PIN-FAULT MODEL	I-31
4.1 Introduction	I-31
4.2 Statistical Pin-Fault Model	I-31
4.3 Derivation of $\phi(z)$	I-36
4.4 Algorithm for the Calculation of $\phi(z)$	I-37
4.5 Computational Results	I-39
REFERENCES	I-47
5.0 SUMMARY OF RESULTS AND FUTURE WORK	I-49
5.1 Summary of Results	I-49
5.2 Future Work	I-49
5.2.1 Theoretical Concept Development	I-50
5.2.2 Software Analyses and Simulation Tools	I-52
5.2.3 Experimental Work	I-52

TABLE OF CONTENTS
(continued)

	<u>Page</u>
APPENDIX A - PROGRAMS FOR COMPUTATION AND SIMULATION	I-57
APPENDIX B - PROGRAM FOR 3-D PLOTS	I-77
APPENDIX C - STATISTICAL FAULT MONITORING OF SEQUENTIAL CIRCUITS	I-81

<u>SECTION II</u> - PROGRAMMABLE CELLULAR ARRAYS WITH HARDWARE ENCODED BUILT-IN-TEST (BIT) FACILITIES		II-1
1.0 INTRODUCTION		II-3
2.0 SEQUENTIAL MACHINE CHARACTERIZATION		II-5
3.0 DESCRIPTION OF BASIC CELL		II-11
4.0 DESCRIPTION OF CELLULAR ARRAY		II-13
4.1 A Procedure for Mapping an Arbitrary Synchronous Sequential Machine Into a Cellular Array		II-13
4.2 Illustration of Mapping Procedure		II-23
4.3 Cellular Arrays with Hardware Encoded BIT Facilities		II-31
5.0 AN ALTERNATE BASIC CELL DESIGN		II-35
6.0 CELLULAR ARRAY IMPLEMENTATION OF A MICROPROCESSOR SYSTEM . . .		II-37
7.0 RESULTS AND CONCLUSIONS		II-39
REFERENCES		II-43
APPENDIX		II-45

<u>SECTION III</u> - MATHEMATICAL MODELS FOR THE DESIGN AND ANALYSIS OF ON-LINE BUILT-IN-TESTING		III-1
1.0 INTRODUCTION		III-3
2.0 ANALYSIS		III-4
2.1 Analysis of a Maintained Module		III-4
2.2 Multiple Fault Classes		III-12
2.3 Multiple Detectors		III-13
2.4 Analysis of a Non-Maintained Module		III-14
3.0 DESIGN		III-17

TABLE OF CONTENTS
(continued)

	<u>Page</u>
<u>SECTION III</u> - MATHEMATICAL MODELS FOR THE DESIGN AND ANALYSIS OF ON-LINE BUILT-IN-TESTING (continued)	
4.0 WORK PLANNED FOR THE FUTURE	III-19
REFERENCES	III-23
 <u>SECTION IV</u> - A FLEXIBLE FAULT-TOLERANT MULTIPROCESSOR ARCHITECTURE FOR VIDEO GRAPHICS	
ABSTRACT	IV-3
1.0 INTRODUCTION	IV-5
2.0 SYSTEM DESCRIPTION	IV-12
3.0 BUILT-IN-TESTING AND FAULT TOLERANCE	IV-28
4.0 OTHER APPLICATIONS	IV-31
5.0 IMPLEMENTATION	IV-31
REFERENCES	IV-33
APPENDIX	IV-35

LIST OF FIGURES

<u>Figure Number</u>	<u>Title</u>	<u>Page</u>
<u>SECTION I</u>		
3.1	Redundant Network for Example 1	I-13
3.2	Markov Model for a JK Flip-Flop	I-16
3.3	A JK Flip-Flop as a Composite Realization	I-17
3.4	4-BIT Shift Register Composition	I-19
3.5	Counter - 74163 Type	I-21
3.6	Output Probability Versus Input Probability for a Synchronous Counter	I-23
3.7	Output Probability with a Failure, C-Stuck-At-One	I-24
3.8	A Module with Output Probability Expressions	I-25
3.9	A General Loop-Free Interconnection of Modules	I-26
3.10	Block Diagram of a System with Intermodule Feedback	I-28
4.1	Model of Synchronous Sequential Circuit	I-32
4.2	Pin-Fault Probability Distribution Example.	I-35
4.3	Logic Diagram of Shift Register	I-41
4.4	P_{esc} , State A	I-43
4.5	P_{esc} , State B	I-44
4.6	P_{esc} , State C	I-45
4.7	P_{esc} , State D	I-46
5.1	Experimental Environment	I-53
<u>SECTION II</u>		
2.1	Basic Cell	II-7
2.2	A Cellular Cascade	II-8
2.3	Encoded Cellular Array	II-9
4.1	Example Array	II-25
4.2	Example of "Hardware Encoded" Cellular Array	II-33
5.1	Alternate Basic Cell	II-36
6.1	A Proposed Cellular Array Organization for a Microprocessor Unit	II-37

LIST OF FIGURES (Continued)

<u>Figure Number</u>	<u>Title</u>	<u>Page</u>
<u>SECTION III</u>		
2.1	Basic System Module	III-5
2.2	A Two-State Model	III-6
2.3	A Four-State Model	III-8
2.4	Apparent Availabilities vs MTDf	III-11
2.5	A State Diagram of System Markov Chain	III-12
2.6	State Diagram	III-13
2.7	Design of a Non-Maintained Module	III-15
3.1	Detector Effectiveness	III-18
3.2	Optimum Check Modulus	III-20
<u>SECTION IV</u>		
1.1	Set of Planar Tiles ("Polygons")	IV-7
1.2	Pipeline Architecture	IV-9
1.3	Buffers	IV-11
1.4	Key Point of Scan Line	IV-13
2.1	Fundamental System Organization	IV-14
2.2	Organization of the Image Buffer	IV-15
2.3	Memory Unit Timing	IV-16
2.4	Organization of Additional Memory Units	IV-17
2.5	Timing Cycles of Additional Memory Units	IV-18
2.6	Memory Address Structure	IV-20
2.7	Unit Interconnection Diagram	IV-21
2.8	Modified Organization	IV-23
2.9	Regular Interlacing for MU and PE	IV-25
2.10	Modification to Increase or Decrease Image Resolution of Processing Speed	IV-26
2.11	Illustration of Physical Organization Corresponding to Various Resolution/Speed Configurations of Figure 2.10	IV-27
<u>APPENDIX</u>		
IV-A.1	Real-Time Image Buffer	IV-39
IV-A.2	Frame Buffer Partitioned into Smaller Units	IV-40
IV-A.3	Obscuring Polygons	IV-42
IV-A.4	Priority Order of Various Image Buffers	IV-43

LIST OF TABLES

<u>Table Number</u>	<u>Title</u>	<u>Page</u>
<u>SECTION I</u>		
2.1	Test Assumptions	I-5
2.2	A Standard BIT Strategy	I-7
3.1	Combinational Gate Probability Models . . .	I-14
3.2	Probabilistic Models for Logic Modules . . .	I-27
4.1	Number of Input Pins	I-35
4.2	State Probabilities	I-42
4.3	Input Probabilities	I-42
5.1	Results Fall 1978	I-49
5.2	Theoretical Issues of Importance to Sampled Monitoring	I-51
5.3	Statistical Experimentation Functional Capabilities	I-54
<u>SECTION II</u>		
4.1	Machine State Transition	II-23
<u>SECTION III</u>		
2.1	Various Combinations of Modulo-m Detectors	III-14

ACKNOWLEDGEMENTS

This report was prepared by the Research Triangle Institute (RTI) in cooperation with North Carolina State University (NCSU), Duke University, and the University of North Carolina (UNC) for the Naval Electronic Systems Command. The RTI project leader was Mr. J. B. Clary under the laboratory supervision of Mr. R. D. Alberts. Dr. J. W. Gault of NCSU, Dr. P. N. Marinos and Dr. Kishor S. Trivedi of Duke, and Dr. H. Fuchs of UNC were the principal researchers responsible for this effort. Sincere appreciation is expressed to Mr. Nathan Butler and Mr. Reeve Peterson of NAVELEX, Code 304, and Mr. William Keiner of the Naval Surface Weapons Center- Dahlgren Laboratories for their guidance and encouragement during the course of this work.

PRECEDING PAGE BLANK-NOT FILMED

SECTION I

SAMPLE FAULT MONITORING (NCSU)

by

James W. Gault
Department of Electrical Engineering
North Carolina State University

January 1979

1.0 INTRODUCTION

The motivation and genesis of this work was developed in earlier reports[1]. Effort during this period has focused upon some rather specific problems and has produced results which have been submitted for publication.

Work is presently focused upon the use of statistical methods for providing on-line monitoring. The philosophy of this approach is given in Section 2. Section 3 defines and describes results which have been obtained in terms of developing the output probability of sequential circuits as a function of input probabilities. Section 4 discusses the evaluation of the effectiveness of monitoring the behavior of output statistics; specifically, the generation of the probability of escape using a statistical pin-fault model. The final section describes the results obtained to date and outlines the direction for future work. Appendices describing software tools developed to date are also included.

2.0 REFERENCELESS ON-LINE MONITORING USING OUTPUT STATISTICS

2.1 Introduction

The work presented here stems from an earlier effort at Research Triangle Institute [2,3] to investigate the feasibility of a standard built-in-test (BIT) circuit suitable for use with a family of digital electronic modules. The search for a standard approach to BIT is motivated by the hope that standardization will make the inclusion of test circuits a more natural and effective part of the system design procedure.

Work described here is presently directed toward statistical procedures and takes as a beginning point the assumptions given in Table 2.1.

Table 2.1 Test Assumptions

1. The testing objective is to detect faults in modular digital electronic equipment which has been fielded, i.e., has passed design and manufacturing acceptance tests.
2. The testing objective is to detect multiple as well as single logic faults, and although it is not clear what the response to intermittent faults will be, they are not specifically excluded.
3. The BIT circuit will monitor on-line operations of a digital module and will not alter normal processing.
4. The behavior of a module being monitored will be characterized statistically assuming stationary and independent input statistics.

2.2 Fault Detection Strategy

Deterministic testing procedures attempt to identify explicit input test vectors which detect the presence of failures by comparing the present network output to the known good result or reference. Previously, manufacturers faced with a large volume of circuits requiring lengthy tests resorted to a technique using random input sequence, an approach which compared the response of a so-called "gold unit" and a unit under test to the same random inputs. This technique led to research directed toward establishing a sounder theoretical basis for probabilistic testing approaches [4-10].

As one would expect, the most substantial results to date are for combinational networks. In particular, precise methods have been defined for deriving the probability that an output is active, as a function of the probabilities for each input, as indicated by Parker and McCluskey [5,6]. Parker and McCluskey have also shown that there exists a set of input probabilities such that no two n -variable combination functions have the same output probability for these inputs. The relevant implication of this result is that a fault-free function may be distinguished from every faulty function if the input probabilities are properly controlled. While some work with sequential networks [9,10] has been reported, no practical methods exist for deriving outputs probabilities as a function of input probabilities. Work is continuing on more effective procedures. The referenceless on-line monitoring strategy proposed here is summarized in Table 2.2.

While there are a number of statistics which may be considered, we are at present only considering the number of active values (ones or zeroes) of a signal over an experiment of length N .

Table 2.2 A Standard BIT Strategy

For a module:

A priori

1. Define a set of points to be monitored. In general, these will include inputs (X), outputs (Z), and internal state values (S).
2. Derive, for the states and outputs, an expected value of the statistics $E_s(x)$ and $E_z(x)$ as a function of the input statistics X .
3. Derive a test stringency ϵ and test length based on a desired test quality, false alarm rate, and escape rate. This will involve the definition of a fault model and the fault density functions $\emptyset(x,z)$, $\emptyset(x,s)$.

On-line

4. During system execution, the fault monitor will:
 - a. Gather statistics on X , S , and Z for an experiment of length N .
 - b. Indicate a failure if a measured statistic (e.g., mz) is outside the acceptance window for the expected value of this statistic, as a function of the measured input statistic mx ; that is, the experiment fails if $mz > EZ(mx) + \epsilon$ or $mz \leq EZ(mx) - \epsilon$ and passes otherwise.

2.3 Evaluating Effectiveness

The approach, as described thus far, holds nondisruptive on-line monitoring as a primary goal. This position does not preclude the active insertion of test vectors during idle periods; however, a totally passive monitor has a number of advantages. For this reason, the effectiveness of passive monitoring will be considered first.

Losq [8] has shown that as a bound the probability of a False Alarm (FA) is given by

$$\text{prob(FA)} \leq \text{erfc} \left(\epsilon \sqrt{2N} \right)$$

This parameter is a function only of the length of the experiment N and the stringency ϵ applied to the measured statistics (m_s, m_z). On the other hand, the probability of Escape (ES) is much more complex and depends upon a fault density function $\emptyset(E_z(x))$ which describes the number of faulty networks which produce $E_z(x) \pm \epsilon$ and, hence, are indistinguishable from the good network. Clearly the probability of escape is strongly related to the input statistics of the current experiment. If the input statistics to a particular module in a particular system and for a particular application are stationary, and if $\emptyset(E_z(x))$ can be computed, then the prob (ES) is fixed for a given N and ϵ . The value of this parameter may be improved by increasing N and decreasing ϵ with the attendant effects on the false alarm rate and time required to detect a failure.

At this point, the desirability of adding an active fault insertion capability to the monitor may be seen by noting that, in general, the prob (ES) may be improved (lowered) by controlling the input probability at a desirable value.

2.4 Status and Conclusions

In order to use a statistical approach to fault monitoring it is necessary to develop a procedure for describing output statistics as a function of the input statistics (e.g., $Ez(x)$). While this is difficult for general sequential circuits, the expressions for sequential primitives (flip-flops, converters, shift registers) have been derived and will be reported elsewhere. Work is continuing on the derivation of module level expressions as a composite of primitive elements.

A more complex problem exists with the derivation of the fault density function $\emptyset(Ez(x))$. While it is possible to develop \emptyset by the enumeration of faulty functions, this is impractical and no suitable analytic procedure has been developed, except for limited cases [8]. This function is essential to the evaluation of the probability of escape. The development of a suitable method for generating the fault density function for a general network is the primary focus of this research. In addition to these primary concerns, this research has also brought into focus the essential need for a fault model which is more useful than the conventional stuck-at-one, stuck-at-zero version. The set of realistic faults for a system, their probability distribution, and the resultant influences on behavior at the network terminals are extremely difficult to define or represent and is essential to meaningful quantitative results.

Another and somewhat newer concern which has surfaced is the need to characterize the input patterns found at module interfaces throughout the system. Is the assumption of stationarity and independence of module input statistics a reasonable position?

The simulation and analytic results obtained to date have provided some encouragement. Statistical monitoring is attractive since its application may be made standard and does not involve either the generation of test vectors or the extensive computation required when a reference is used. At the present time the primary drawbacks to statistical monitoring are the lack of computationally feasible procedures for output and density functions, as well as the lack of understanding of the behavior of faults and input patterns which strongly influence the effectiveness of the approach.

REFERENCES

1. Clary, J. B., J. W. Gault, P. N. Marinos, K. S. Trivedi and D. L. Parnas, "Basic Research in Support of Concurrent Fault Monitoring in Modular Digital Systems," Interim Report, Contract N00039-77-C-0363, June 1978.
2. Clary, J. B., Gault, J. W., Weikel, S. J., Whisnant, R. A., Alberts, R. D., A Study of a Standard BIT Circuit, Final Report, Contract No. 0163-76-C-0231, February 1977.
3. Gault, J. W. and Clary, J. B., "The Application of Minicomputers as On-line Built-In-Test Elements", Proceedings of IEEE Southeastcon, April 1978.
4. Parker, C. P., "Compact Testing: Testing with Compressed Data", Proceeding FTCS-76, June 1976.
5. Parker, C. P. and McCluskey, E. J., "Analysis of Logic Circuits with Faults Using Input Signal Probabilities", IEEE Transactions on Computers, May 1975, pp. 573-578.
6. Parker, C. P. and McCluskey, E. J., "Probabilistic Treatment of General Combinational Networks", IEEE Transactions on Computers, June 1975, pp. 668-670.
7. David, P. and Blanchet, G., "About Random Fault Detection of Combinational Networks", IEEE Transactions on Computers, June 1976, pp. 659-664.
8. Losq J., "Referenceless Random Testing", Proceedings of the FTCS-76, June 1976.
9. Shedletsky, J. and McCluskey, E. J., "The Error Latency of a Fault in a Sequential Digital Circuit", IEEE Transactions on Computers, June 1976, pp. 655-659.
10. Parker, K. P., and McCluskey, E. J., "Sequential Circuit Output Probabilities from Regular Expressions", IEEE Transactions on Computers, March 1978, pp. 222-231.

3.0 SEQUENTIAL CIRCUIT OUTPUT PROBABILITIES FROM NETWORK REALIZATIONS

3.1 Introduction

The idea of using random sequences as test patterns for digital electronic networks was born out of a practical need to cut the cost of developing deterministic test sequences. This approach involves driving a reference unit, which is assumed to be fault-free, and a unit-under-test with the same random sequence, while comparing their respective outputs. The practicality and effectiveness of using random test sequences have been treated extensively in the literature [1,2]. This concept has evolved and now there is considerable interest in a more general view of non-deterministic testing.

Non-deterministic testing involves:

- 1) the description of a network input stream as distributions of a set of independent random variable(s), and
- 2) the characterization of the distributions of a set of dependent random variables representing the response of a fault-free network to these inputs.

Underlying this strategy is the hypothesis that the presence of a fault will alter the distribution of the response sufficiently to detect its presence.

Present accomplishments fall significantly short of this general problem objective. The intention of this paper is to pursue a more limited non-deterministic testing goal which is: Given the description of a realization of a network as an interconnection of more primitive sub-modules (e.g., as found in any typical design automation system); derive the expression for the probability that an output is equal to one as a function of the probability that the primary inputs are equal to one.

The papers which are central to the development of this work will be summarized to provide the essential background. Even a stopped clock is correct twice a day and, similarly, a digital signal has some probability of being correct, defined even in the presence of faults. Ogus [3] uses this notion and develops procedures by which signal probabilistic models for primitive logic elements. This concept was extended by Parker and McCluskey [4,5] and

procedures were given for the derivation of output probabilities for general combination networks. Parker and McCluskey had shown earlier [6] that there exists a set of input probabilities x_1, x_2, \dots, x_n given by $x_i = \frac{1}{2^{2^i + 1}}$ such that all n variable combination functions have distinct values of output probabilities. This result substantiates the underlying hypothesis that faults in combinational networks will perturb output probabilities. No such result exists for sequential networks.

Far less has been done to develop similar results for sequential networks. The important concept of error latency, which is applicable to combination circuits as well, was presented by Shedletsky and McCluskey [7]. Their error latency of a fault is the number of input vectors applied to a network with that fault until the first incorrect output vector is observed.

Work specifically concerned with the output probability behavior of sequential circuits has proceeded from behavioral models. Markov chains can be defined from state tables or diagrams [9] and Parker and McCluskey have demonstrated the derivation from regular expression descriptions [8].

We will develop procedures which utilize network realizations rather than behavior models. This point of view is motivated by the belief that the probabilistic test strategy will be a portion of a design automation system.

The next section considers the generation of output probabilities for certain common functions which will then be treated as library descriptions for derivations involving more complex composite networks. General composite networks are then treated in Section 3.

3.2 Output Probabilities for Common Functions

In this section we will consider the output probability expressions for:

- 1) general combinational networks,
- 2) flip-flops,
- 3) n -bit shift registers, and
- 4) n -bit binary modulo 2^n counters.

Methods have already been presented for handling general combinational networks [4,5]. Method 1 of reference [5] uses the minterm canonical description of a function as the model for derivation and hence is a behavioral approach. A second method is given which allows a function to be modeled by an equation in any desired form; thus, it is suitable for treating realizations.

All inputs and gate outputs are assigned a unique symbol and the input variables are considered to be mutually independent statistically. The output probability is derived by moving from primary inputs to outputs using the first five basic properties given in Table 3.1, which is an update to Table 3.1 in reference [5]. Upper case letters are used for variables and lower case for the probability that the variable is true. The development in [5] makes use of an unstated conjecture that if the primary input variables are independent, then all internal variables are likewise independent, even at reconvergent fanout points. This notion is also used here along with the additional observation that, to insure a valid output probability expression for a combinational network, the expression must be written in a sum-of-products form.

Example 1 Consider the combinational module of Figure 3.1. The development of the output expression is given by

$$d = bc,$$

$$e = 1 - (1-b)(1-c) = b + c - bc,$$

$$g = de \text{ if } d, e \text{ are independent and by substitution}$$

$$g = bc(b+c-bc), \quad (1)$$

$$g = b^2c + bc^2 - b^2c^2 \quad (2)$$

$$\text{and finally by table 1 [4,5]}$$

$$g = bc. \quad (3)$$

Clearly the probability value obtained for g will be different if equation 1 or 2 is used rather than equation 3. It can be stated then that a valid output probability expression or a general combination network may be obtained from the sum-of-products form of the probability equation which was derived by Method 2 [5]. It should also be noted that the network of Example 1 is redundant and the sum-of-products expression obtained is still valid; hence, no special processing is required to remove redundancy from a design file.

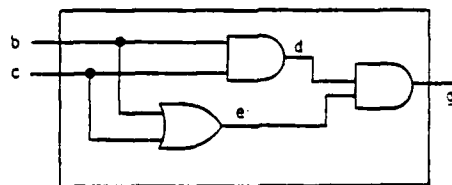


Figure 3.1 Redundant Network for Example 1

Table 3.1 Combinational Gate Probability Models

FUNCTION	PROBABILITY	REMARKS
1) $F = \bar{x}_i$	$f = 1 - x_i$	
2) $F = x_1 \cdot x_2 \cdots x_n$	$f = x_1 \cdot x_2 \cdots x_n$	where all x_i mutually independent
3) $F = x_1 + x_2 + \cdots x_n$ is considered as $F = \overline{x_1 \cdot x_2 \cdots x_n}$	$f = 1 - \prod_{i=1}^n (1-x_i)$	where all x_i mutually independent
4)	$f = x_i \cdot x_i = x_i^*$ and in general $\prod_{i=1}^n x_i$ or $x_i^n = x_i$	where all x_i are associated with the same time slot; used when f is combinational.
5)	$f = x_i + x_i = 2x_i$ and in general $f = \sum_{i=1}^n x_i = nx_i$	independent of time slot; used when f is combinational or sequential
6)	$f = x_i(t_j) \cdot x_i(t_k) = x_i^2 t_j t_k$ and in general $f = \prod_{i=1}^n x_i = x^n$	where x_i are from distinct time slots; used when f is sequential.

*Generalized in Table 2.

Next let us consider the derivation of steady state output probabilities for a simple sequential network, a JK flip-flop. We will first derive this expression using the familiar Markov chain methods and then we will develop the same expression considering the flip-flop to be a realization composed of simpler primitives.

Example 2 [10] Figure 3.2 shows the Markov model, transition matrix for a JK flip-flop.

The steady state probabilities are

$$P_A = \frac{j}{j+k} \quad \text{the probability that } Q=1 \quad (4)$$

$$P_B = \frac{k}{j+k} \quad \text{the probability that } Q=0. \quad (5)$$

The same result can be obtained by solving the equation set

$$\begin{aligned} P_A &= jP_B + (1-k)P_A \\ P_B &= (1-j)P_B + kP_A \\ P_A + P_B &= 1. \end{aligned} \quad (6)$$

Example 3 [10] A gate realization for a JK flip-flop is given in Figure 3.3a. The approach will be to derive an output probability expression for the combinational network and then use that as a library function in the derivation for the composite network of Figure 3.3b. Thus,

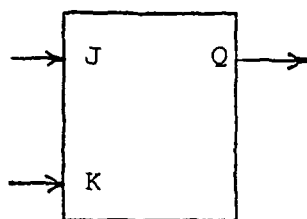
$$y^* = J \cdot \bar{y} + \bar{K} \cdot y \quad (7)$$

which, treated as a combinational function and utilizing Table 3.1(1,2,3), gives

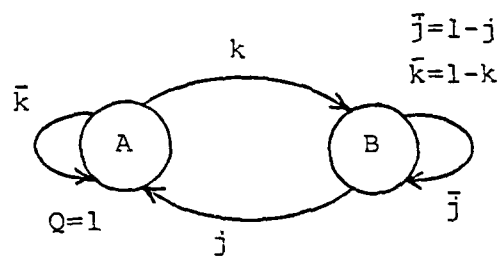
$$y^* = 1 - (1-(j) (1-y)) (1- (1-k) (y)) \quad (8)$$

which in sum-of-products is

$$y^* = j+y-jy-ky \quad (9)$$



a) JK flip-flop



b) Markov model

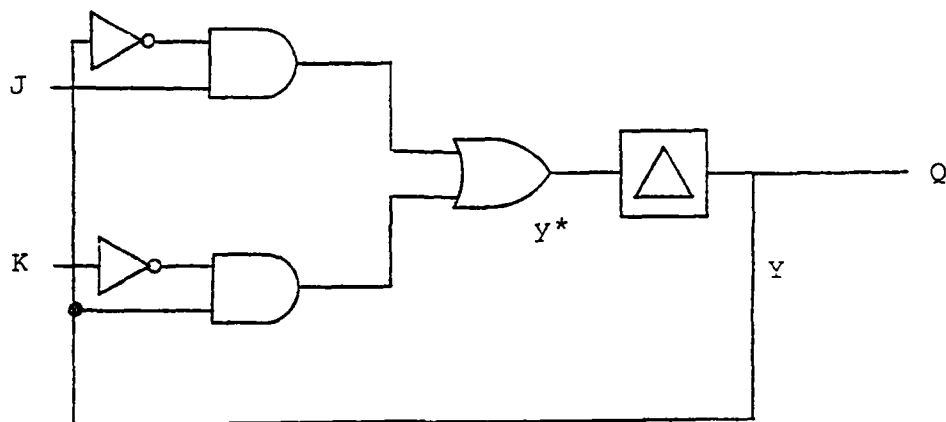
$$P = \begin{matrix} & \begin{matrix} A & B \end{matrix} \\ \begin{matrix} A \\ B \end{matrix} & \begin{bmatrix} 1-k & k \\ j & 1-j \end{bmatrix} \end{matrix}$$

c) transition matrix

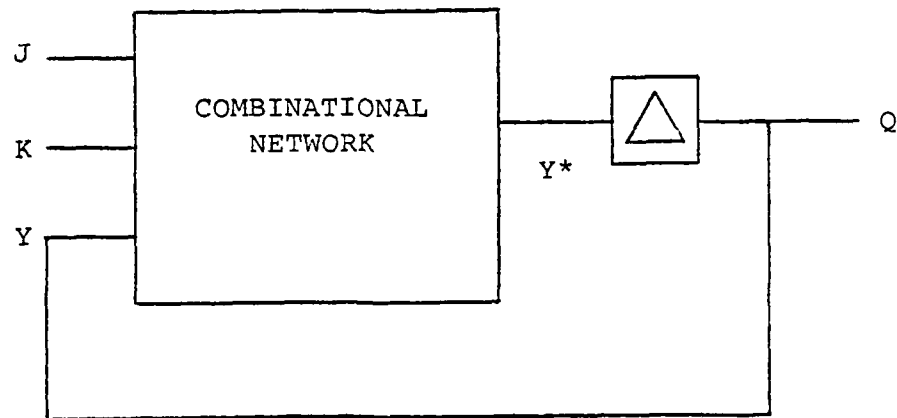
$$\lim_{n \rightarrow \infty} P^n = \begin{matrix} & \begin{matrix} A & B \end{matrix} \\ \begin{matrix} A \\ B \end{matrix} & \begin{bmatrix} \frac{j}{j+k} & \frac{k}{j+k} \\ \frac{j}{j+k} & \frac{k}{j+k} \end{bmatrix} \end{matrix}$$

d) steady state transitions

Figure 3.2 Markov Model for a JK Flip-Flop



a) JK flip-flop realization



b) block diagram of composite

Figure 3.3 A JK Flip-Flop as a Composite Realization

Now, consider the composition of Figure 3.3b.

$$Q(t) = Y^*(t-1) \quad (10)$$

and

$$q(t) = y^*(t-1) \quad (11)$$

When Equation (9) is substituted into Equation (11)

$$q(t) = j(t-1) + y(t-1) - j(t-1)y(t-1) - k(t-1)y(t-1)$$

and

$$q(t) - y(t-1) = j(t-1) - y(t-1)(j(t-1) + k(t-1)) \quad (12)$$

which by Table 1 (5) and $q(t) = y(t)$ reduces to

$$y(t-1) = \frac{j(t-1)}{j(t-1) + k(t-1)} \quad (13)$$

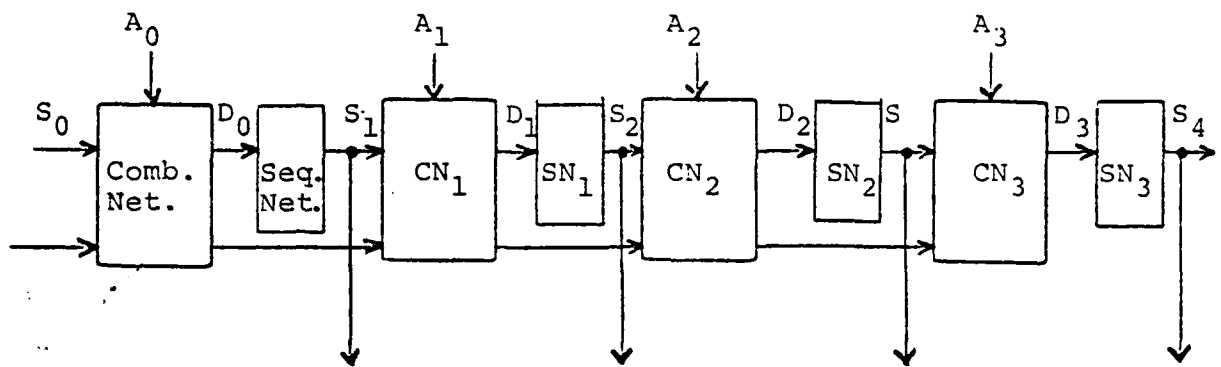
and in the steady state is

$$y = q = \frac{j}{j+k} \quad (14)$$

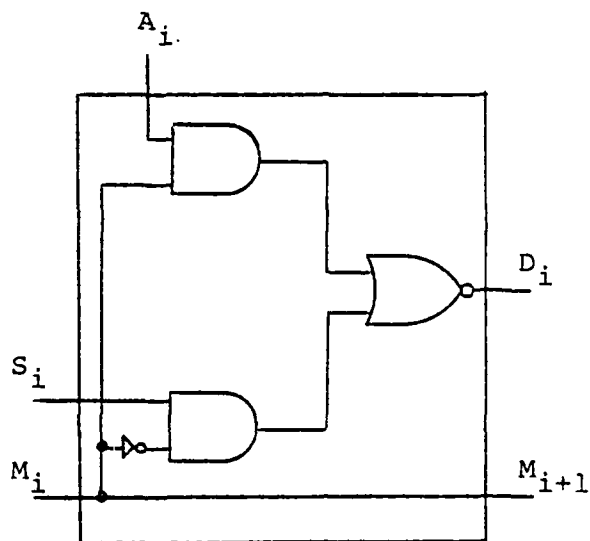
which is in agreement with the Markov development (4) since $PA = q$.

This derivation may be questioned since y^* , from the combinational block (9), was derived under the assumption that J , K , and Y are independent. The fact that Y is made equal to $Y^*(t-1)$ through feedback makes this assumption questionable. Notice, however, that at any point in time $J(t)$, $K(t)$ and $Y(Y^*(J(t-1), K(t-1)))$. That is, Y is independent of J, K if $J(t-1), K(t-1)$ are independent of $J(t), K(t)$. The general applicability of this strategy will evolve throughout this paper.

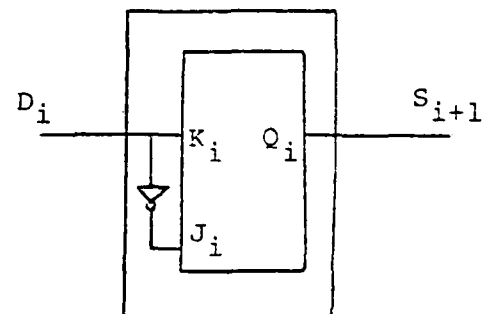
Example 4 Consider the collection of combination and sequential blocks of Figure 3.4, which is a 4-bit shift register similar to a 7495 TTL device.



a) block diagram of shift register composition



b) combinational network realization



c) sequential network realization

Figure 3.4 4-Bit Shift Register Composition

The output probability expression for the combinational block is given by

$$d_i = 1 - s_i \bar{m}_i + a_i m_i \quad (14)$$

where $m=1$ is a load and m is a shift.

Using the previous result for a JK flip-flop

$$q_i = \frac{j_i}{j_i + k_i} \quad \text{and} \quad j_i = 1 - d_i, \quad K_i = d_i$$

Therefore,

$$s_{i+1} = q_i = 1 - d_i \quad (15)$$

and

$$s_{i+1} = s_i \bar{m}_i - a_i m_i \quad (16)$$

The probability that a particular output is one in the steady state may be computed recursively. For example,

$$s_3 = s_0 \bar{m}_0 \bar{m}_1 \bar{m}_2 + a_0 m_0 \bar{m}_1 \bar{m}_2 + a_1 m_1 \bar{m}_2 + a_2 m_2. \quad (17)$$

Property 6 of Table 1 applies and since all m are of equal value

$$s_3 = s_0 \bar{m}^3 + a_0 m \bar{m}^2 + a_1 m \bar{m} + a_2 m, \quad (18)$$

This expression may be verified using Markov analysis. Generalizing this expression we obtain

$$s_i = s_0 \bar{m}^i + \sum_{j=0}^{i-1} a_j m \bar{m}^{(i-j-1)}. \quad (19)$$

If the probability of obtaining a particular shift value is desired (for example, 0100), it is written as

$$\bar{s}_1 s_2 \bar{s}_3 \bar{s}_4 = (1-s_1) s_2 (1-s_3) (1-s_4). \quad (20)$$

The general expression for a shift register output bit, given by Equation (19), can now be used as a library function for future compositions.

Example 5 In the manner of Example (4), a counter (Figure 3.5) may be characterized by defining the various modes as

$$\begin{array}{lll} R = \text{low, reset} & \alpha = \bar{P}\bar{T} \cdot \bar{L} \cdot \bar{R} & \lambda = \frac{\beta}{1-\alpha} \\ PT = \text{high, count enable} & \beta = PT \cdot \bar{L} \cdot \bar{R} & \\ L = \text{low, parallel load} & \gamma = L \cdot \bar{R} & \end{array}$$

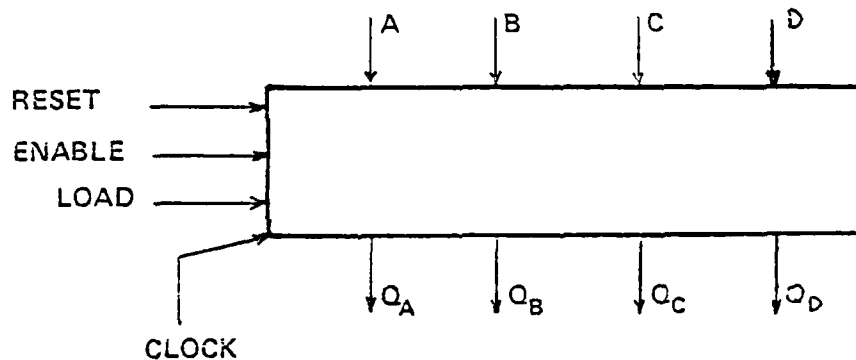


Figure 3.5 Counter - 74163 Type

which results in the probability of any particular output Q_i being given recursively as

$$q_i = \frac{\gamma d_i + \beta h_i}{2\gamma d_i + 2\beta h_i + r} \quad (21)$$

where d_i is the probability of a 1 at the input to bit i and

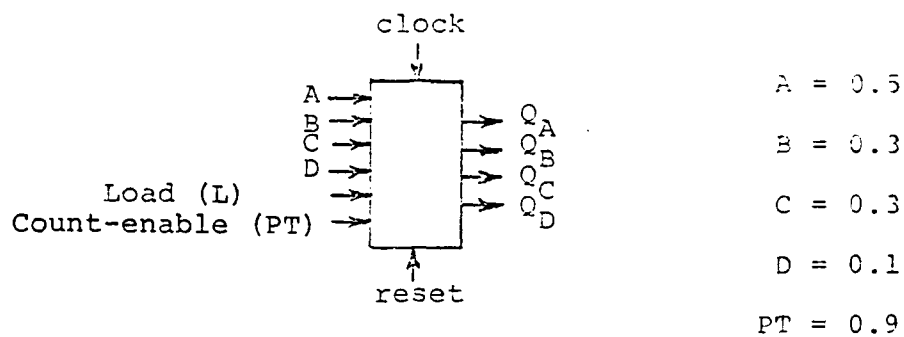
$$\begin{aligned} h_i &= \prod_{j=0}^{i-1} q_j \quad i > 1 \\ h_0 &= 1 \end{aligned} \quad (22)$$

which can, in turn, be used to derive the following expressions for total state values (count value)

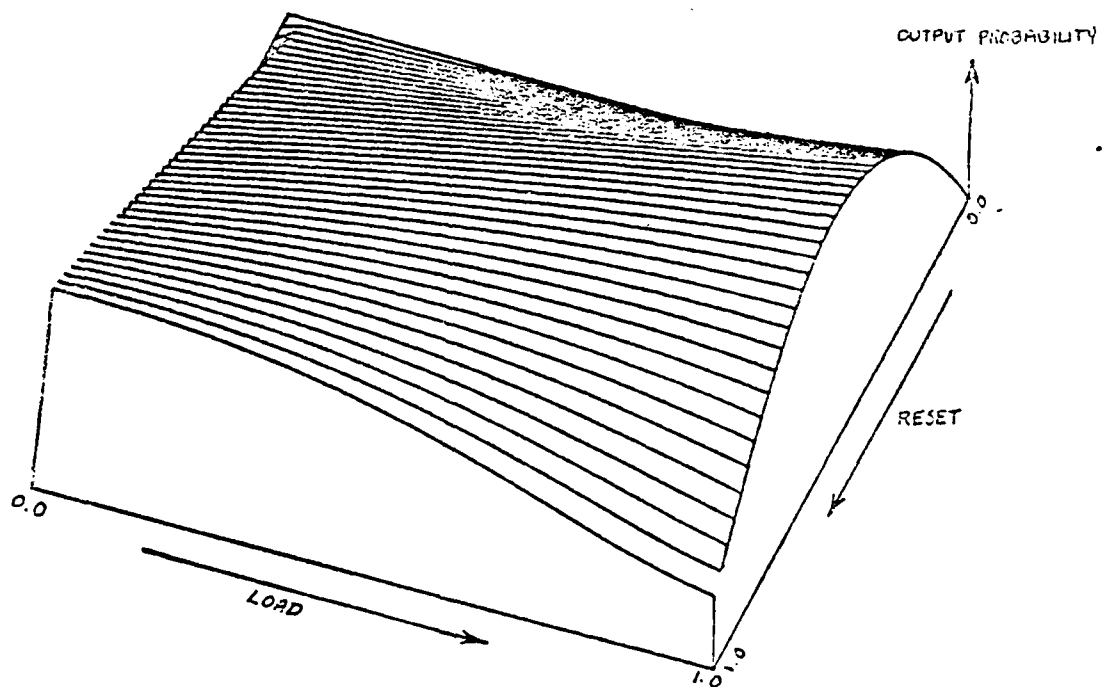
$$S_0 = \frac{1 - \frac{\gamma}{1-\alpha} \left(\frac{1}{1-\alpha} \right) \sum_{k=1}^{N-1} i_k (1 - \lambda^{N-k+1})}{1 + \sum_{n=1}^N \lambda^n} \quad (23)$$

$$S_j = \lambda^j S_0 + \frac{\gamma}{1-\alpha} \sum_{k=1}^N \lambda^{j-k} i_k \quad j=1,2,\dots,N \quad (24)$$

where i_k is the probability of a parallel load value of K . For example, if $k=3$, then for a four-bit counter $i_3 = \bar{d}_3 \bar{d}_2 d_1 d_0$. The applicability of these derivations is demonstrated by Figures 3.6 and 3.7. These curves were generated for the counter of Example 5 using a set of assumed probabilities as noted. The surface describing the probability of being in state s_1 is given in Figure 3.6 as a function of the LOAD and RESET probabilities. If the package input associated with C is stuck-at-one, then this surface is altered as shown in Figure 3.7. Clearly, the ability to detect this fault is dependent upon the module input statistics [10,11,12].

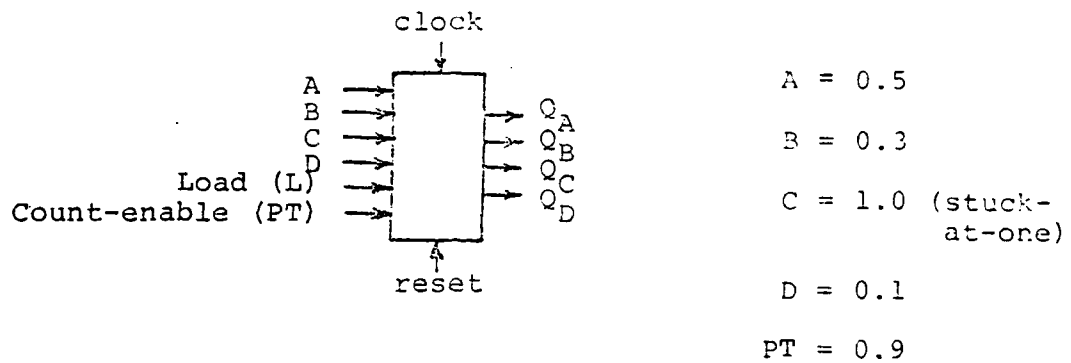


a. counter package and constant input probabilities

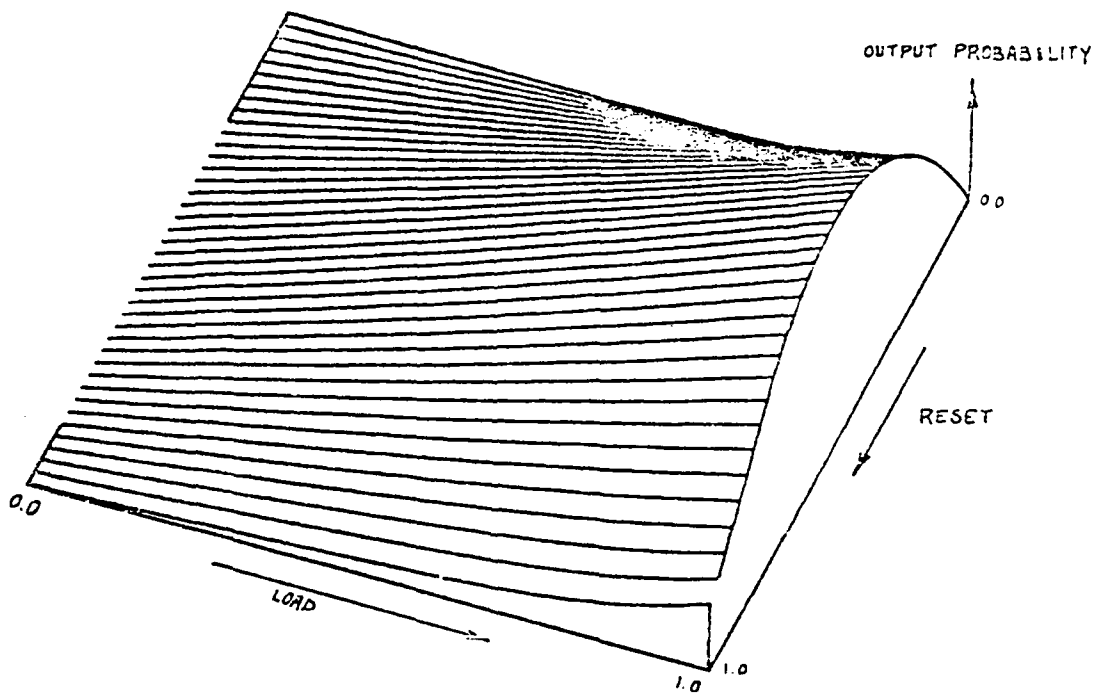


b. probability of S_1 vs. LOAD and RESET

Figure 3.6 Output Probability Versus Input Probability for a Synchronous Counter



a. counter package and constant input probabilities



b. probability of S_1 vs. LOAD and RESET

Figure 3.7 Output Probability with a Failure, C-Stuck-At-One

3.3 Output Probabilities for General Composite Networks

The objective of this section is to clarify the general applicability of the ideas exemplified in the last section. The approach taken to this point has been to use the Markov model to validate expressions derived using network realizations. General rigorous formalisms have not yet been developed; however, there are useful operational properties which may be tentatively identified.

Consider a complex system to be made up of an interconnection of modules (Figure 3.8), whose output probability expression has been tabularized in a library. Any output of such a module may in general be either combinational or sequential.

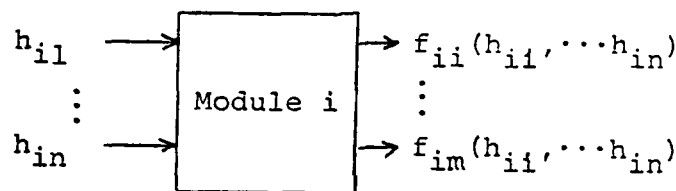


Figure 3.8 A Module with Output Probability Expressions

We will first look at systems which are loop-free, as in Figure 3.9. There is no feedback in the module interconnections. In processing a network the objective is to generate probability expressions

$$z_i(\underline{x}) \quad \forall z_i \in \underline{z}$$

proceeding from inputs to outputs, making substitutions in the appropriate $f_{ij}(\underline{h})$ library functions. Properties 1 and 2 of Table 2 describe the forms of f_{ij} and h_i at individual modules.

In evaluating a network from primary inputs \underline{x} to primary outputs \underline{z} at each module, a substitution of the previously generated $h_i(\underline{x})$ value is made in $f_{ij}(\underline{h})$. This substitution may result in product terms involving a single primary input variable with various degrees of exponentiation. These cases are handled differently for combinational or sequential $f_{ij}(\underline{h})$ as shown by Properties 3 and 4 of Table 3.2. When making a substitution into a sequential

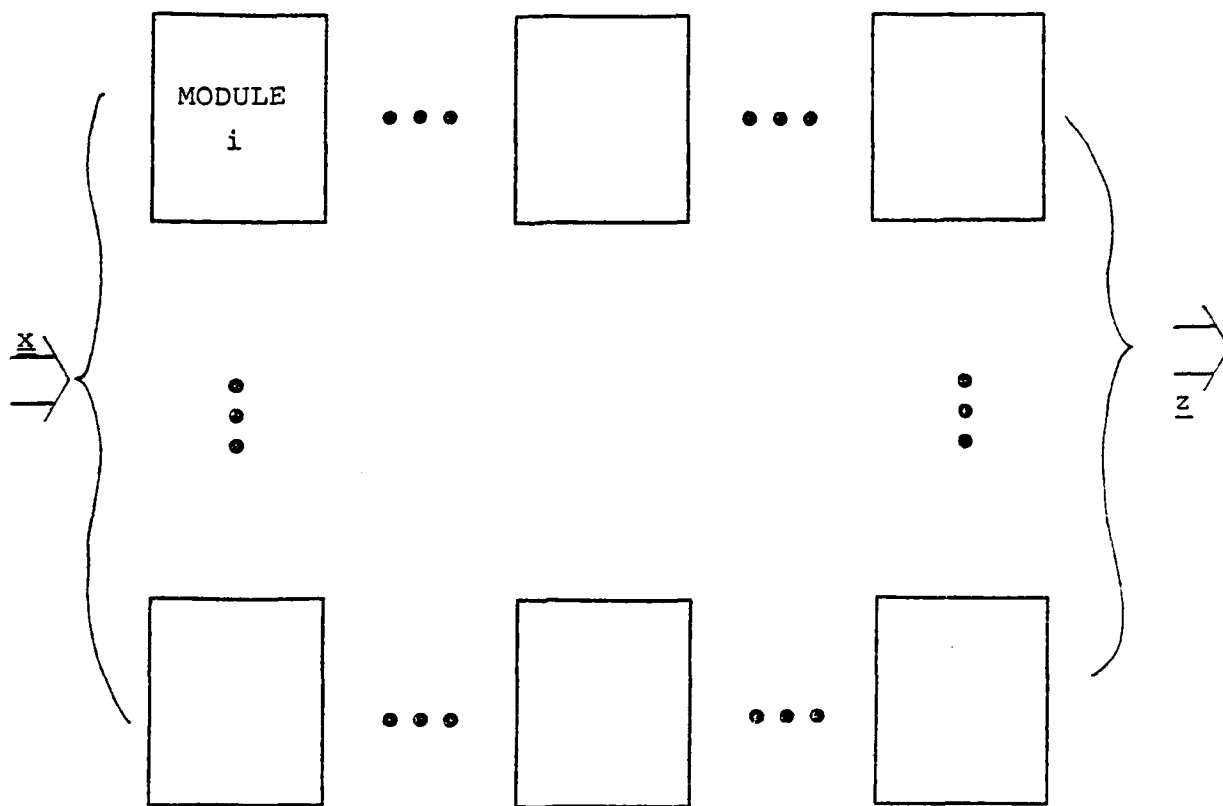


Figure 3.9 A General Loop-Free Interconnection of Modules

Table 3.2 Probabilistic Models for Logic Modules

PROBABILITY EXPRESSION		REMARKS
1.	$f_{ij}(\underline{h})$	Is exponent free when f_{ij} is a combinational function. May include h_i terms if f_{ij} is sequential.
2.	$h_i(\underline{x})$	Will be exponent free when all successors to h_i are combinational. May include x_i terms if <u>any</u> successor <u>is</u> sequential.
3.	for $f_{ij}(\underline{h})$ when substitution in $h_i(\underline{x}) \cdot h_j(\underline{x})$ yields $x^i \cdot x^j = x^k$, then $k = \min(i, j)$	When f_{ij} is combinational. This is a generalization of Table 1 (4).
4.	for f_{ij} when substitution in $h_i(\underline{x}) \cdot h_j(\underline{x})$ yields $x^i \cdot x^j = x^k$, then $k = i + j$	When f_{ij} is sequential. This is a generalization of Table 1 (6).
5.	for f_{ij} when substitution in $(h_i(\underline{x}))^j$ yields $(x^i)^j = x^k$, then $k = i \cdot j$	When f_{ij} is sequential. Will not occur when f_{ij} is combinational.
6.	Additive or subtractive probabilities follow normal laws of algebra for combinational or sequential $f_{ij}(\underline{h})$.	

$f_{ij}(\underline{h})$, an exponentiation of an exponented variable may result and is treated as indicated by Property 5, Table 3.2.

At the present time, formal proofs for these properties have not been generated. They stand as conjectures which have been utilized successfully in a number of examples.

Structures involving intermodule feedback may be modeled in the classic form shown in Figure 3.10. M_1 and M_2 may be compositions of other modules and will, in general, be sequential. The resolution of this structure follows the form given by

$$\underline{z}(\underline{x}) = (\underline{x}, \underline{y}(\underline{y}^*(\underline{x}))). \quad (25)$$

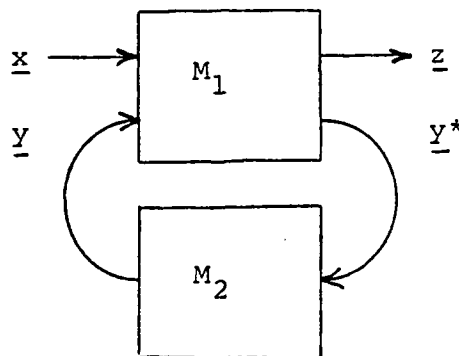


Figure 3.10 Block Diagram Of a System with Intermodule Feedback

The derivation of the desired expression of output probabilities \underline{z} as functions of only primary input probabilities \underline{x} requires the solution of a set of simultaneous, nonlinear equations. Only preliminary work has been done with this class of networks.

3.4 Summary

The testing of digital networks by examining statistics of behavior rather than absolute behavior is attractive for a number of reasons. In order to apply this approach it is necessary to be able to describe the expected values of a set of dependent variables as functions of the expected values of the appropriate independent variables. At the present time the statistic which is most often used is the number of occurrences of an active signal on a line during a specified time period. This event may be stated as the probability of occurrence of a signal over an experiment of N time trials. In this paper methods are described for computing the expected value of this output probability as a function of input signal probabilities. This derivation requires that the network be described as a realization, that is, a design file description of the interconnection of modules, which are library functions whose probability behavior has been catalogued.

We have demonstrated the approach for some specific examples and in the process have generated some useful library descriptions. In addition, the properties (Tables 3.1 and 3.2) which are useful in generating composite expressions for larger networks are given as conjectures.

Considerable effort and future research will be required to produce the rigorous proofs which are essential before these ideas can be widely used. Effort is continuing in that direction.

REFERENCES

1. Parker, C. P., "Compact Testing: Testing with Compressed Data", Proc. of the Int'l. Conf. on Fault-Tolerant Computing (FTCS-6), June 1976, pp. 93-98.
2. Hayes, J. P., "Transition Count Testing of Combinational Logic Circuits", IEEE Trans. on Computers, Vol. C-25, No. 6, June 1976, pp. 613-620.
3. Ogus, R. C., "The Probability of a Correct Output From a Combinational Circuit", IEEE Trans. on Computers, Vol. C-24, No. 5, May 1971, pp. 534-544.
4. Parker, K. P., and McCluskey, E. J., "Analysis of Logic Circuits with Faults Using Input Signal Probabilities", IEEE Trans. on Computers, Vol. C-24, No. 5, May 1975, pp. 573-578.
5. _____, "Probabilistic Treatment of General Combinational Networks", IEEE Trans. on Computers, Vol. C-24, No. 6, June 1975, pp. 668-670.
6. _____, "Boolean Network Probabilities and Network Design", Technical Note No. 60, Digital Systems Lab, Stanford University, July 1975.
7. Shedletsky, J. J. and McCluskey, E. J., "The Error Latency of a Fault in a Sequential Digital Circuit", IEEE Trans. on Computers, Vol. C-25, No. 6, June 1976, pp. 655-659.
8. Parker, K. P. and McCluskey, E. J., "Sequential Circuit Output Probabilities From Regular Expressions", IEEE Trans. on Computers, Vol. C-27, No. 3, March 1978, pp. 222-231.
9. Isaacson, D. L. and Madsen, R. W., Markov Chains: Theory and Applications, Wiley and Sons, 1976.
10. Ardalan, S. H., "Statistical Fault Monitoring of Sequential Circuits", Master of Science Thesis, Dept. of Elec. Engr., NC State University, December 1978.
11. Gault, J. W. and Ardalan, S. H., "Referenceless On-Line Monitoring", Proc. of the Government Microcircuits Applications Conf., November 1978.
12. Gault, J. W., et al., "Sample Fault Monitoring", Basic Research in Support of Concurrent Fault Monitoring in Modular Digital Systems, Interim technical report, Research Triangle Institute, RTP, NC, June 1978.

4.0 A STATISTICAL PIN-FAULT MODEL

4.1 Introduction

Primary to the evaluation of the probability of escape and therefore to the determination of effectiveness of a given test is the derivation of the fault density function. Specifically, the probability of escape is related to the fault density function $\phi(z)$ [1] according to the expression

$$\text{prob}(\text{pass test/faulty}) = \sum_{K=N(z_j-\epsilon)}^{N(z_j+\epsilon)} \int_0^1 \binom{N}{K} z^K (1-z)^{N-K} \phi(z) dz \quad (1)$$

where N and ϵ are the test length and stringency, respectively. The function $\phi(z)$, which depends on the input statistics, is derived for a given circuit under a specified fault model. In Section 2 we developed a statistical fault model under the pin-fault assumption proposed by Ketelsen [2]. Section 3 showed how the fault density function for a module is obtained from the statistical fault model and knowledge of the output probability expression. An algorithm for computing the fault density function will be given in this Section [3]. This algorithm will be used to compute the escape probabilities of an integrated sequential circuit example in Section 5.

4.2 Statistical Pin-Fault Model

An appropriate fault model for fielded digital electronic equipment is the pin-fault model proposed and justified by Ketelson [2]. This model incorporates faults occurring due to "lead bonds" failures of input or output transistors, or failures due to improper connection of the package pins to the silicon chip, for example. According to this model, the input and output pins of integrated circuits experience stuck-at-one or stuck-at-zero faults.

In this section a statistical pin-fault model will be developed for digital synchronous sequential circuits. The abstract model of a synchronous sequential circuit is shown in Figure 4.1. In this model inputs occur at discrete intervals of time and each application of inputs results in, at most, one-state transition. The delay elements are implemented using suitable flip-flops (master-slave or edge triggered) to insure proper synchronous operation.

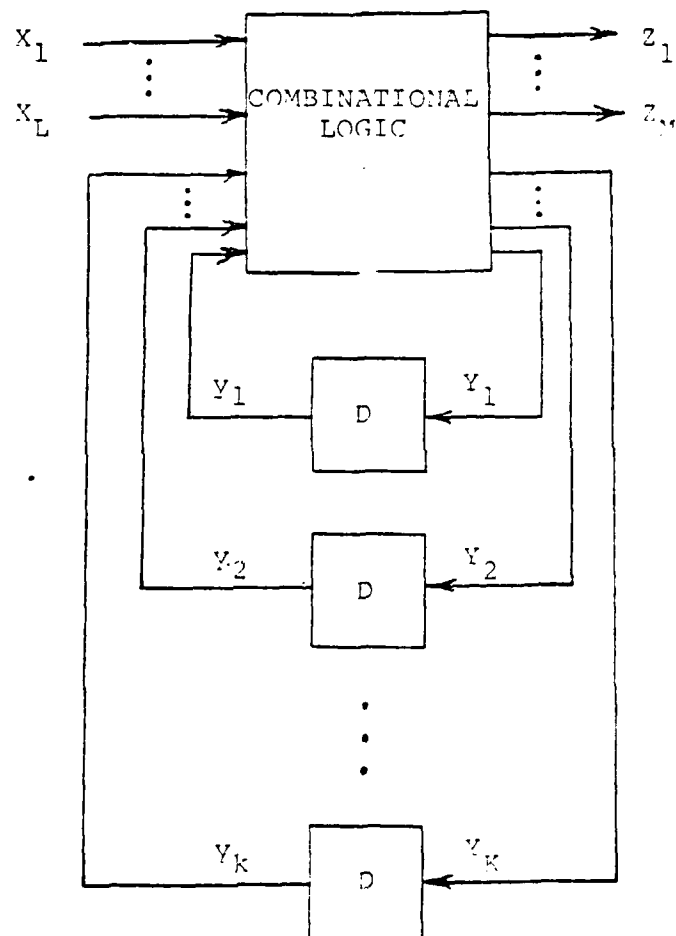


Figure 4.1 Model of Synchronous Sequential Circuit

We will now examine the model in order to determine, in an abstract way, the effect of various faults on the statistics of the outputs. Referring to Figure 4.1, stuck-at-faults at the outputs y_j of the delay elements cause the number of states to decrease. This causes significant deviation of the faulty circuit statistic from the fault-free statistic, particularly if the outputs are the internal states of the machine. This deviation of output statistics, due to faults on the delay element outputs, is thus immediately detectable. The same arguments can be stated for faults on the D-type delay element input lines. If the delay elements are implemented with JK flip-flops, the nature of the effects of faults on the input lines of the delay elements is more complicated. Faults on the inputs of set-reset flip-flops, however, cause the outputs to be stuck-at-one or stuck-at-zero and are therefore detectable, as in the case of D-type flip-flops.

Unlike the faults mentioned above, faults occurring on the input lines x_k to the sequential circuit may cause minor changes in the output statistics. For example, a failure may change the transition from state 0_i to 0_j under input A_k ($0 \leq A_k \leq 2^L - 1$). According to Losq [1], "these failures will be far more difficult to detect because their effect on the system operation is somewhat limited (in a statistical sense)". Faults occurring in the combinational logic also may cause minor changes in output statistics.

Therefore, since faults occurring at the input pins to the sequential circuit dominate other faults, we propose a fault model that considers faults, we propose a fault model that considers faults to occur only at the input pins of the sequential circuit. Hence, if the test strategy is capable of detecting faults at the input pins, faults at the delay elements and output pins are also detected. This fault model is known as the pin-package fault model. Note that we exclude faults occurring in the combinational logic, since we are dealing with fielded integrated circuits. Faults occurring in the combinational logic have a very low probability of occurrence compared to faults at the input pins (i.e., for a silicon chip the probability that a single gate function fails is very low for fielded integrated circuits). However, an advantage of statistical testing of the outputs is that, although knowledge of the effect of faults in the combinational circuit on the output is difficult to analyze, it can be shown that most faults cause deviations from the fault-free statistics and are thus detectable. The exclusion of the analyses of these faults in the fault model

greatly simplifies the analyses of the problem and makes the systematic analyses of the test strategy possible while at the same time taking into account a large percentage of the high probability faults.

In our discussion below, we will be concerned with integrated circuit implementations of the model in Figure 4.1. In this case, we only have access to the input and output pins of the circuit. Thus, in the pin-fault model faults in the form of stuck-at-one or stuck-at-zero faults occur only on the input and output pins. We place no restriction on the number of pins that can be at fault.

The total number of faulty input combinations, assuming stuck-at-one and stuck-at-zero faults at the pins of an input circuit, is

$$N_f = 3^L - 1 \quad (2)$$

If we let k denote the number of pins at fault, then the number of possible combinations of k stuck-at faults is

$$n_f = 2^K \binom{L}{k} \quad (3)$$

We assign a probability P_k to the probability that k pins are at fault simultaneously. We then have

$$\sum_{k=1}^L 2^K \binom{L}{k} P_k = 1 \quad (4)$$

It is reasonable to assume that the probability of occurrence of multiple pin-faults is a decreasing function of the number of faulty pins; that is, P_k decreases as k increases. In this paper we use the model

$$P_k = \epsilon^{-\alpha k} \quad (5)$$

However, any other model can be employed. Thus, in the statistical fault model considered, the probability of faults at the input pins decreases exponentially as the number of simultaneous faulty pins increases. The exponent coefficient α corresponding to the number of input pins L is listed in Table 4.1.

Table 4.1 Number of Input Pins

Number of input pins		λ
1	0.6931472	
2	1.574520	
3	2.040524	
4	2.358062	
5	2.598983	
6	2.793101	
7	2.955652	
8	3.095474	
9	3.218129	
10	3.32738	
11	3.425883	
12	3.515547	
13	3.597834	
14	3.673863	
15	3.744524	
16	3.810523	

Let the variable f be an integer defined in the interval $1 \leq f \leq 3^L - 1$. Then each integer f defines a specific fault configuration with a corresponding number K simultaneous pin-faults. We thus define the function $g(f)$ as the probability of the occurrence of the faulty configuration f .

This function will be used in the derivation of the fault density function in the next section. The variable f can be ordered such that the number of faulty pins specified by faulty configuration f increases as f increases. The function $g(f)$ plotted against such an ordering of f is shown in Figure 4.2.

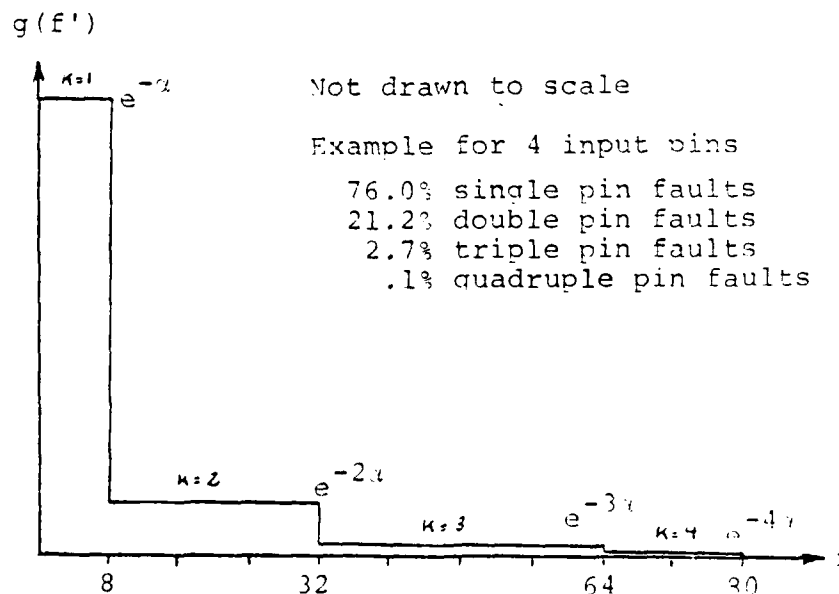


Figure 4.2 Pin-Fault Probability Distribution Example

4.3 Derivation of $\phi(z)$

In this section the fault density function will be derived for a synchronous, sequential integrated circuit using the statistical pin-fault model. Let the function relating the output probabilities to the input probabilities be $Z=T(x_1, x_2, \dots, x_\ell)$. Under the pin-fault assumption, only stuck-at-zero or -one faults occur at the input pins. Thus, when input line x_i , $i=1, 2, \dots$ is at fault, the faulty output statistic is either

$$Z_f = T(x_1, x_2, \dots, x_{i-1}, 0, \dots, x_\ell) \quad (6)$$

or

$$Z_f = T(x_1, x_2, \dots, x_{i-1}, 1, \dots, x_\ell) \quad (7)$$

depending on whether the pin is stuck-at-zero or stuck-at-one.

Again, let the integer f be in the region $1 \leq f \leq 3^\ell - 1$. Then each integer f defines a faulty configuration and, as will be shown in the next section, the corresponding faulty output statistic, Z_f , as a function of the integer f , assuming fixed input statistics, can be expressed as

$$Z_f = T(f) \quad (8)$$

where $T(f)$ is related to $T(x_1, x_2, \dots, x_\ell)$, the fault-free transfer function.

The fault density function $\phi(z)$ is the density of fault circuits that have been output statistic z . We can thus arrive at an expression for $\phi(z)$ in terms of the functions $Z_f = T(f)$ and $g(f)$.

This expression is given in (10). If we define

$$\delta[Z - T(f)] = \begin{cases} 1.0 & \text{if } Z = T(f), \text{ i.e., } z = Z_f \\ 0.0 & \text{if } Z \neq T(f). \end{cases} \quad (9)$$

then

$$\phi(z) = \sum_{f=1}^{3^{\ell}-1} \delta[z-T(f)]g(f) \quad (10)$$

As a check to the above derivation, we note that $\int_0^1 \phi(z) dz \equiv 1$. Applying this relation to (10) we have

$$\int_0^1 \phi(z) dz = \int_0^1 \sum_{f=1}^{3^{\ell}-1} \delta[z-T(f)]g(f) dz = \sum_{f=1}^{3^{\ell}-1} g(f) \int_0^1 \delta[z-T(f)] dz \quad (11)$$

$$= \sum_{f=1}^{3^{\ell}-1} g(f) = 1.$$

The following section presents an algorithm for computing $\phi(z)$, i.e., evaluating Equation (10) on a digital computer. This procedure is then used to calculate accurately the escape probability for digital sequential integrated circuits in a defined test.

4.4 Algorithm for the Calculation of $\phi(z)$

The mathematical discussion in Section 3 suggested an algorithm for the computation of the fault density function. We shall first attempt to develop a method for obtaining a fault configuration (ordered in increasing number of faulty pins).

The total number of faulty configurations is $N=3^{np}-1$. Where np is the number of input pins. We define the integer f in the region $0 \leq f \leq N$. Expanding f in powers of 3 we obtain

$$f = \alpha_1 + 3\alpha_2 + 3^2\alpha_3 + \dots + 3^{np-1}\alpha_{np} \quad (12)$$

where $\alpha_i = 0, 1$, or 2 . Thus, each integer f is defined by a set $[\alpha_1, \alpha_2, \dots, \alpha_{np}]$. We shall define $\alpha_i = 2$ to mean that pin i is fault-free and $\alpha_i = 0$ or 1 to mean pin i is stuck-at-zero or -one, respectively. Thus, if $np=4$, the set $[2102]$ representing $f=2+3+0+27 \times 2=59$ means pin numbers one and four are fault-free, pin

number two is stuck-at-one, and pin number three is stuck-at-zero.

We can obtain the faulty output statistic Z_f corresponding to a faulty configuration $[\alpha_1, \alpha_2, \dots, \alpha_{np}]$ by defining coefficients β_i according to (11) and (12) and relating the coefficients β_i to α_i .

$$Z_f = T(\beta_1 X_1, \beta_2 X_2, \dots, \beta_i X_i, \dots, \beta_{np} X_{np}) \quad (13)$$

$$\beta_i X_i = \begin{array}{ll} X_i & \text{pin } i \text{ is fault-free } (\alpha_i=2) \\ 0 & \text{pin } i \text{ is s-a-0 } (\alpha_i=0) \\ 1 & \text{pin } i \text{ is s-a-1 } (\alpha_i=1) \end{array} \quad (14)$$

From Equation (14) we observe that

$$\beta_i = \begin{array}{ll} 0 & \alpha_i = 0 \\ 1/x_i & \alpha_i = 1 \\ 1 & \alpha_i = 2. \end{array} \quad (15)$$

We thus obtain a relationship between β_i and α_i which can be expressed as

$$\beta_i = \frac{\alpha_i}{2} [1 + (\alpha_i - 2) \lambda_i] \quad (16)$$

which satisfies Equation (15) if $\lambda_i = \frac{x_i - 2}{x_i}$. Substituting for λ_i , we obtain the equation

$$\beta_i = \frac{\alpha_i}{2} [1 + (\alpha_i - 2) \frac{x_i - 2}{x_i}] \quad (17)$$

relating β_i to α_i .

We are now in a position to present the algorithm for computing $\phi(z)$. The first step is to generate the coefficients α_i in the order of increasing number of faulty pins. For each set of coefficients, there is a corresponding probability P_k defined by the relation $P_k = e^{-\alpha k}$, where k is the number of faulty pins in the faulty configuration defined by the coefficients. Thus, as the coefficients are generated, the probability density function $g(f)$ of faulty configurations f is also generated. From Equation (17) the coefficients β_i are derived. Using these coefficients, the output probability Z_f corresponding to the faulty configuration f is obtained using Equation (13). Finally, as the functions $g(f)$ and $Z_f = T(f)$ are obtained through this procedure, the fault density function is calculated using the relationship of Equation (10). The above algorithm has been implemented in FORTRAN.

We have thus developed a fault model which is computationally feasible and which has a reasonable coverage of high probability faults. In addition, we have shown how the fault density function $\phi(z)$ can be derived using this model. The final step of using ϕ to compute the probability of escape is treated in the next section.

4.5 Computational Results

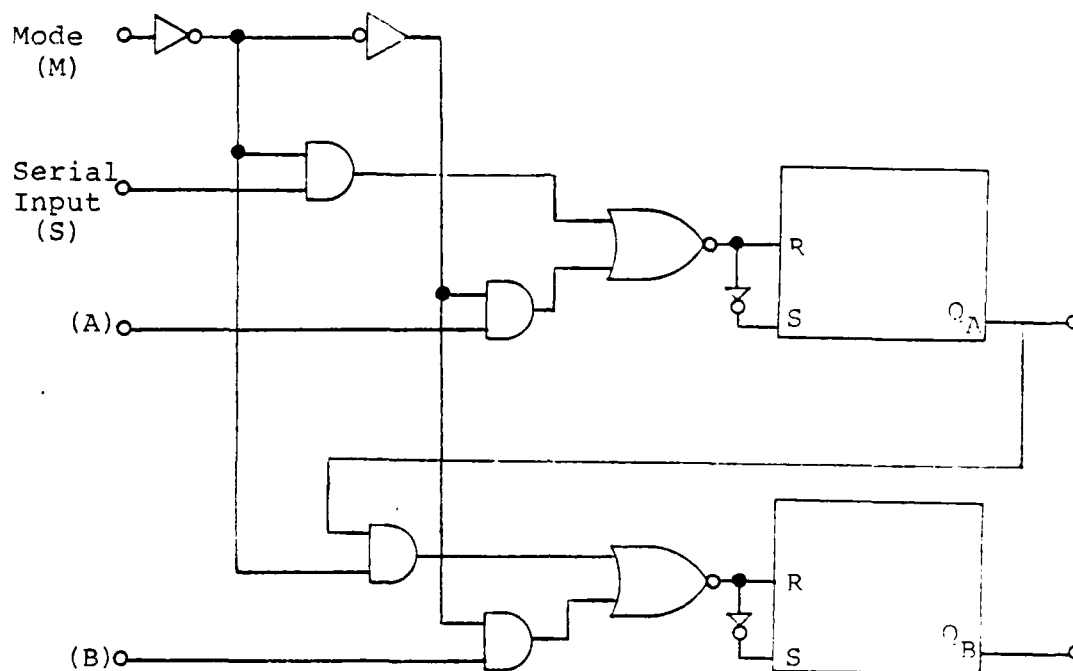
In this section the probability of escape is calculated and results are presented for a typical synchronous sequential integrated circuit. The computations are done using the algorithm described in the previous section. The results are verified by computer simulations of the devices under various faulty configurations. In the simulations, pseudorandom inputs are applied and the statistics of the monitored outputs of the device are collected. If the output statistics are within the specified range of the fault-free statistics, the devices pass the test. Thus, the criterion for passing a simulated test can be summarized by

$$N(Z_i - \epsilon) \leq Z_c \leq N(Z_i + \epsilon) \quad (18)$$

where Z_c is the number of occurrences of a one at the output and N , ϵ , and Z_i are the number of simulations (test length), test strigency, and fault-free statistic, respectively. It may be required that all outputs satisfy the above condition.

The logic diagram of a two-bit, right-shift parallel-load shift register is shown in Figure 4.3. The relationships between output and input probabilities (steady state) are given in Table 4.2. Using these relationships and the statistical pin-fault model, the probability of escape was obtained for each output. These results were based on specific input probabilities given in Table 4.3 and a test length $N=10,000$ and test stringency $\epsilon=0.01$. In obtaining the results, the statistics of one pin was varied between zero and one, while the statistics of the remaining pins were held constant. Figures 4.4 through 4.7 show that very good testing regions exist where the probability of escape is extremely low. For instance, with the input statistics given in Table 4.3 the probability of escape by monitoring state B is $P_{esc}=2.58 \times 10^{-11}$. These results suggest that statistical methods are useful in testing this type of circuit. In fact, if the input statistics are controllable and we use the plots obtained, tests resulting in extremely low escape probabilities can be designed ($P_{esc}=5.3 \times 10^{-17}$ for $m=0.8$, $s=0.5$, $a=0.3$, and $b=0.2$ by monitoring state A, $N=10,000$, $\epsilon=0.01$). However, in an on-line environment where prior knowledge of the input statistics exists and are uncontrollable, using the computational procedures developed, the escape probability and therefore the test efficiency can be evaluated for a given experiment [4]. The plots obtained for the probability of escape show that if the inputs exhibit random behavior in an on-line situation, regions exist for which statistical testing of circuits is quite attractive (i.e., low probability of escape).

In addition, the results show that in some regions where the probability of escape is high for some states, it is low for others. This suggests monitoring the statistics of a number of states (outputs) simultaneously to achieve a low escape probability over a wide region of input statistic variation. For example, by monitoring both states A and B, the escape probability will be low, as the statistics of pin B in the range $0.1 \leq b \leq 0.9$.



Input Pins	Output Pins
Mode (M)	
Serial input (S)	Q_A
Parallel input (A)	Q_B
Parallel input (B)	

Figure 4.3 Logic Diagram of Shift Register

Table 4.2 State Probabilities

State	Output Probability Expression
A (00)	$(\bar{s} \bar{m} - s \bar{s} \bar{m}^2 - \bar{s} a m \bar{m} + \bar{b} \bar{a} m)$
B (01)	$(s \bar{m} - s^2 \bar{m}^2 - s a m \bar{m} + \bar{b} a m)$
C (11)	$s^2 m^2 + a s \bar{m} m + m a b$
D (10)	$\bar{m}^2 s \bar{s} + \bar{m} m \bar{s} a + b \bar{a} m$
$\bar{m} = 1-m$	m prob. of mode input
$\bar{s} = 1-s$	s prob. of serial input
$\bar{a} = 1-a$	a prob. of parallel load input A
$\bar{b} = 1-b$	b prob. of parallel load input B

Table 4.3 Input Probabilities

Input Pin	Probability
Mode (M)	0.8
Serial in (S)	0.5
Parallel in (A)	0.3
Parallel in (B)	0.6

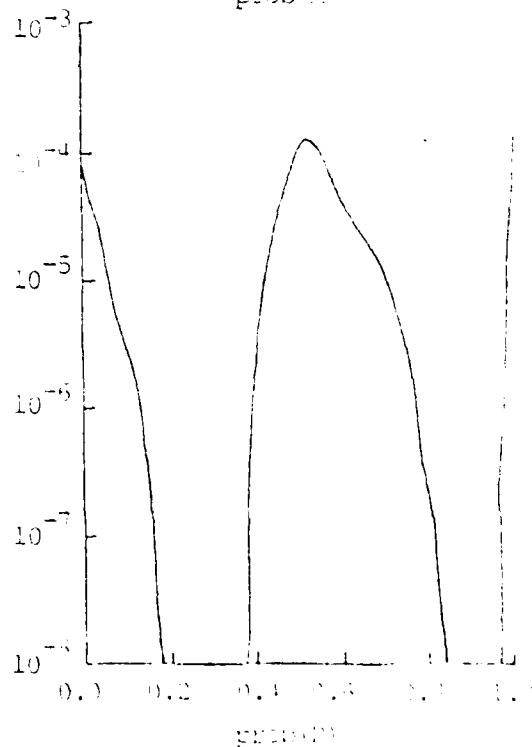
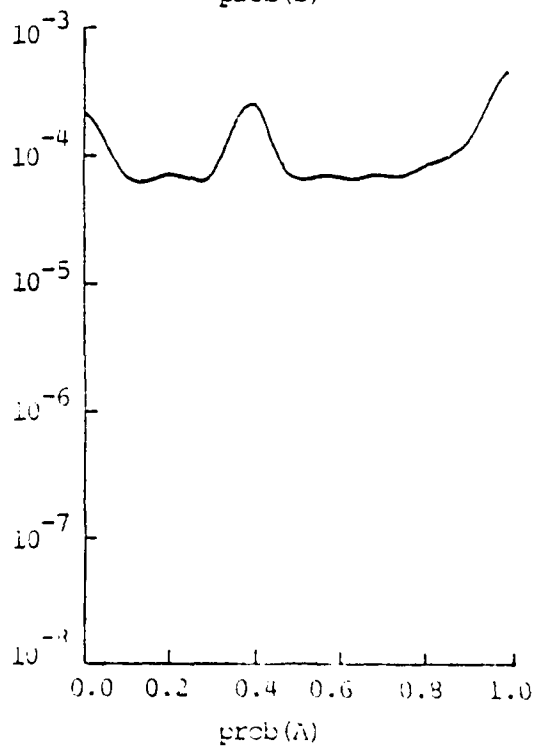
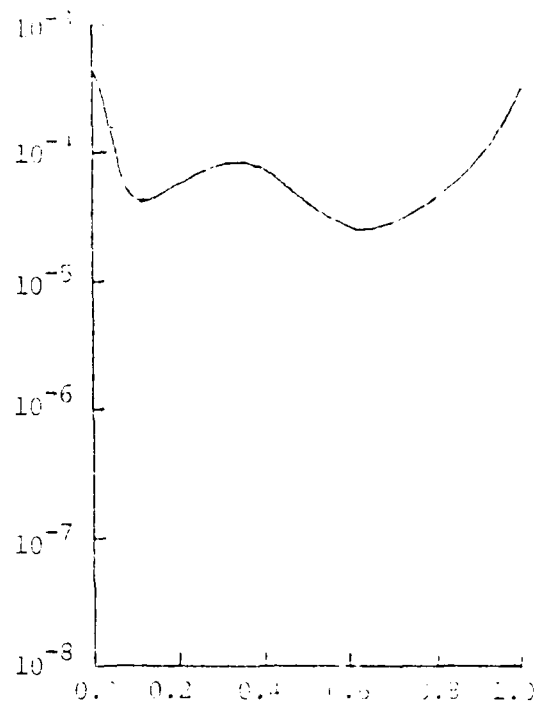
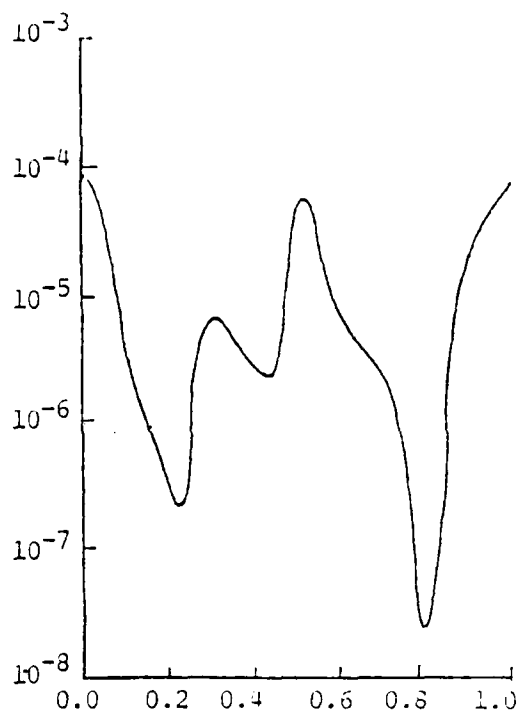


Figure 4.4 P_{esc} , State A

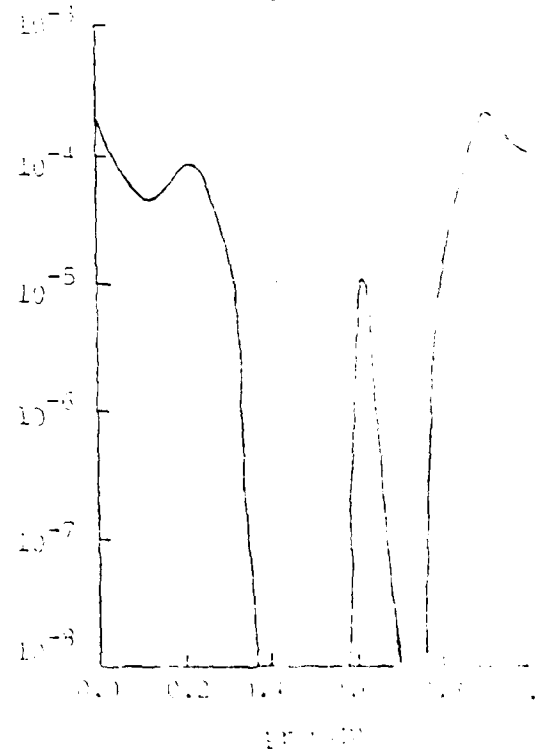
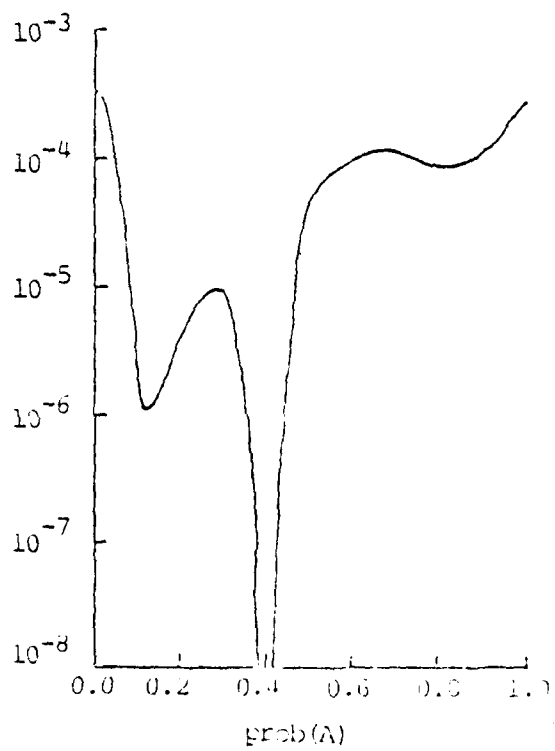
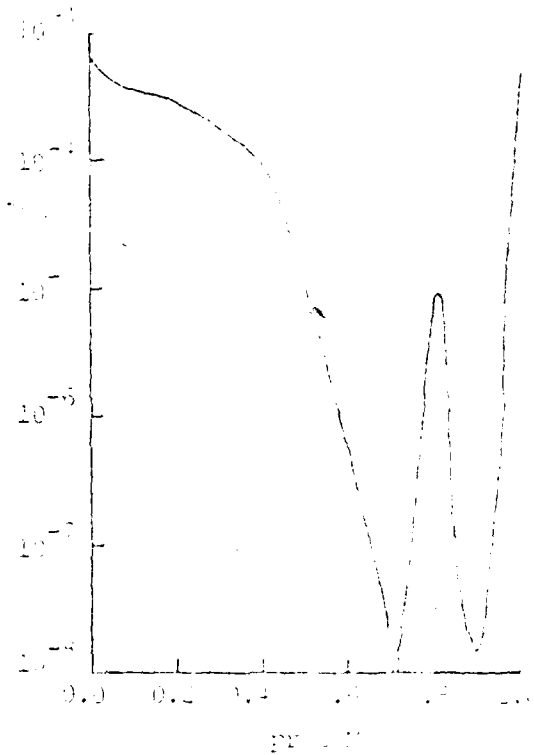
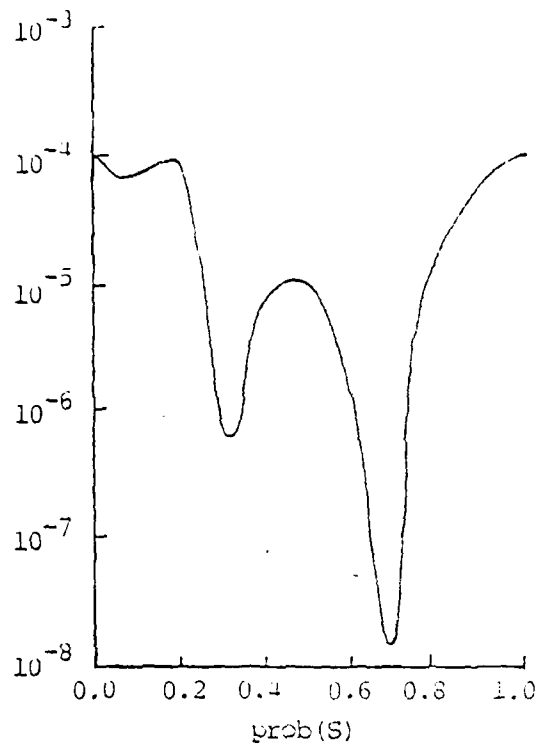


Figure 4.5 P_{esc} , State B

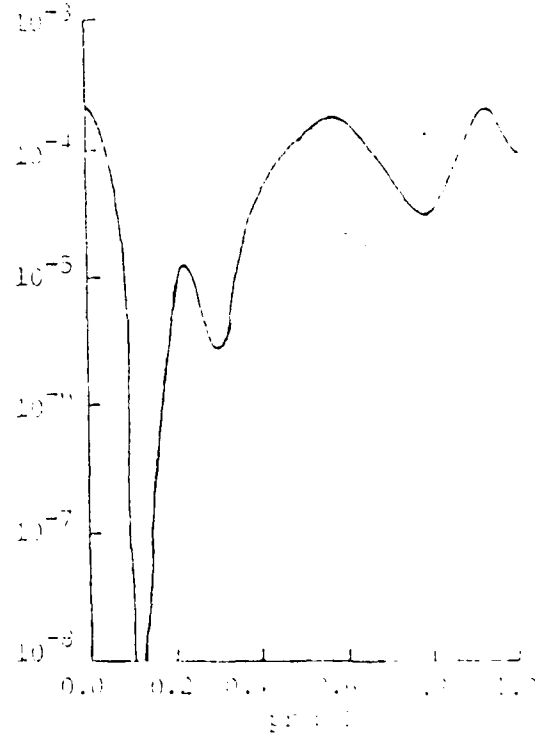
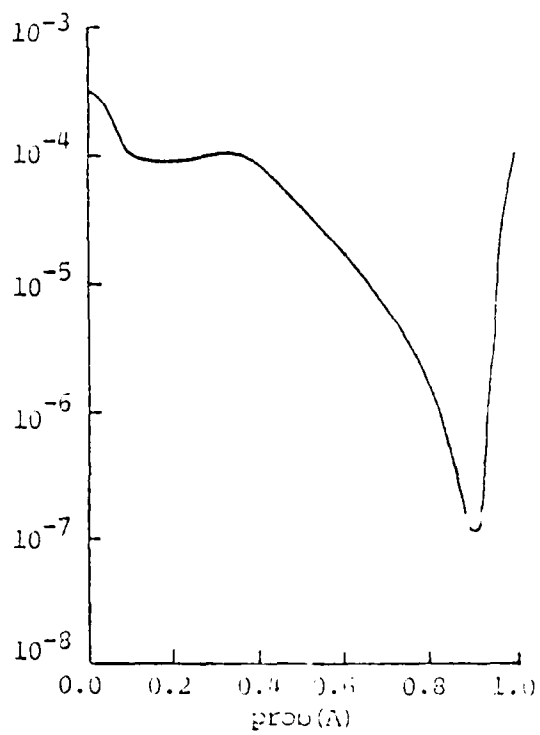
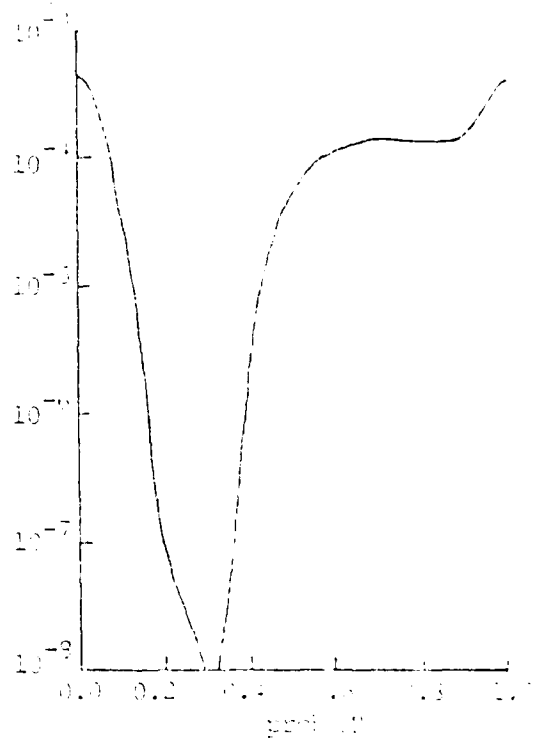
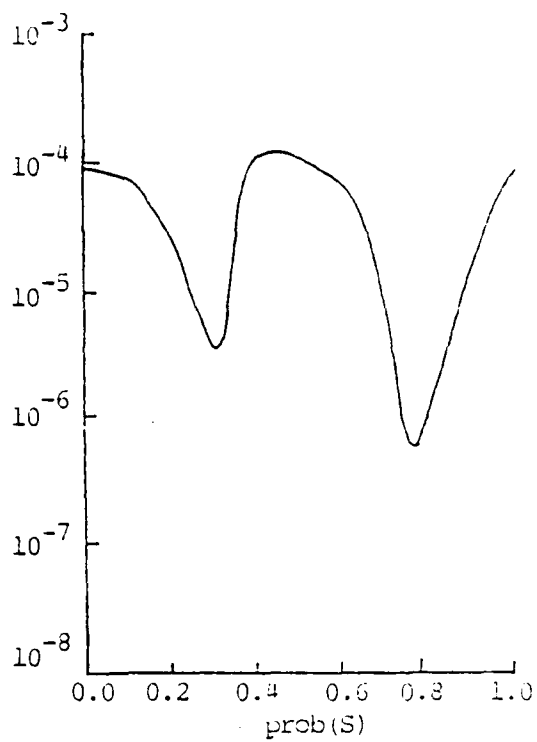


Figure 4.6 P_{esc} , State 0

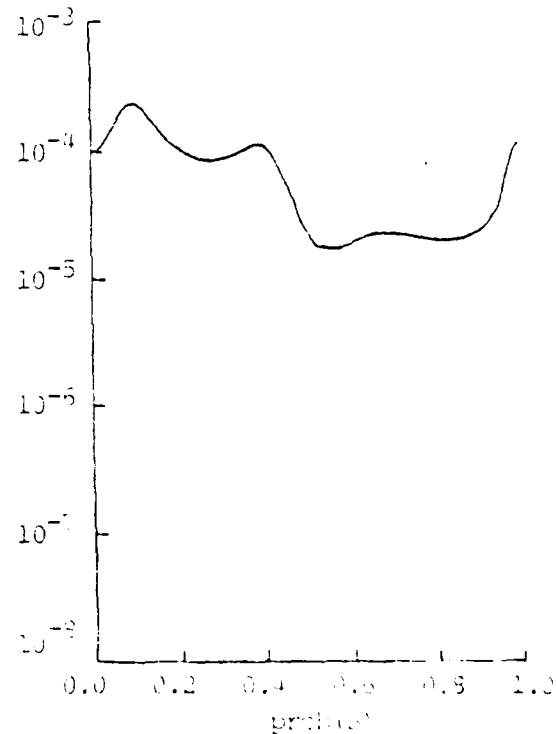
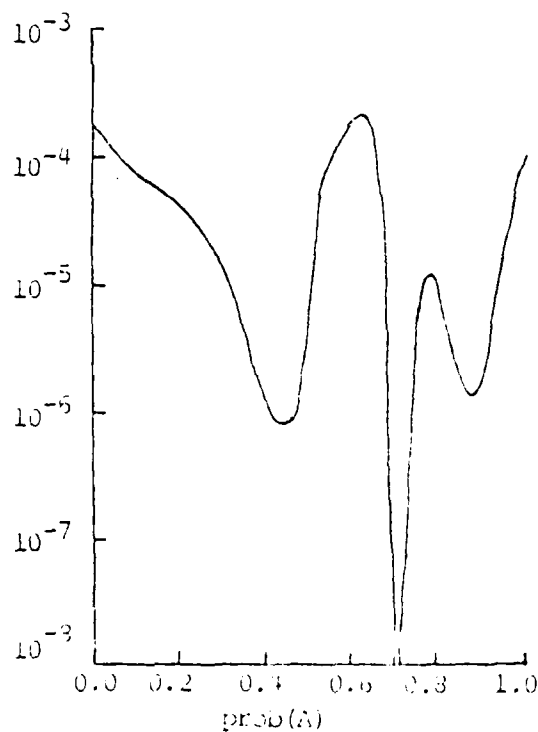
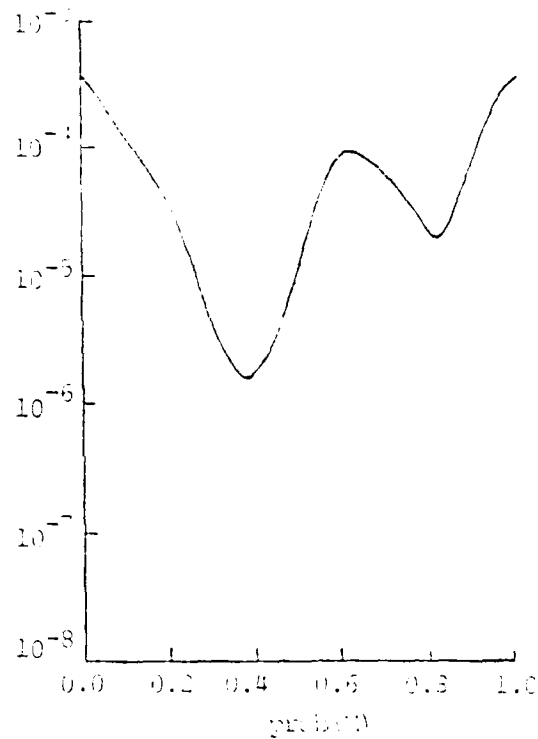
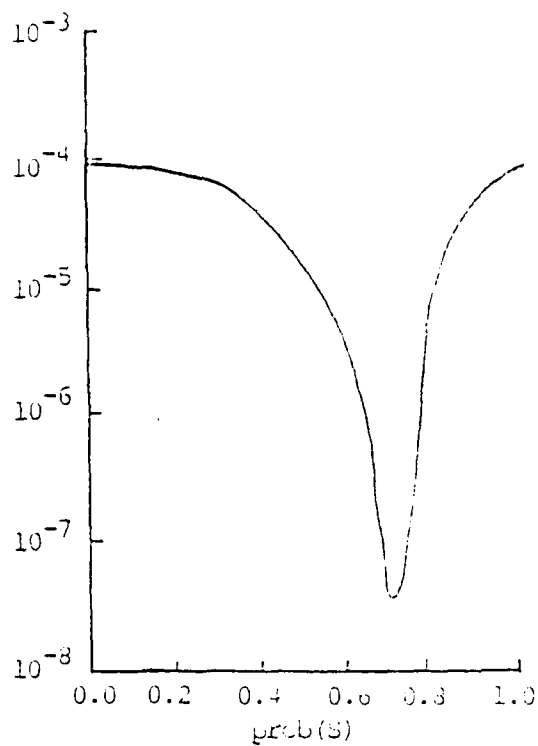


Figure 4.7 $\text{prob}(S)$, State D

REFERENCES

1. Losq, J. "Referenceless Random Testing", Proc. of Int'l. Symposium on Fault-Tolerant Computing, 1976, pp. 108-113.
2. Ketelsen, M. L., An Integrated Circuit Fault Model for Digital Systems, PhD Dissertation, Report R-743, Coordinated Science Laboratory, University of Illinois, Urbana, Illinois, 1976.
3. Ardalan, S. H., Statistical Fault Monitoring of Sequential Circuits, Master's Thesis, Dept. of Electrical Engineering, NC State University, Raleigh, NC, December 1978.
4. Gault, J. W. and Ardalan, S. H., "Referenceless On-Line Monitoring", Proc. of the Government Microcircuits Applications Conference, November 1978.

5.0 SUMMARY OF RESULTS AND FUTURE WORK

5.1 Summary of Results

Results during this last period have culminated in the papers listed in Table 5.1. In addition, preliminary work has begun on the development of software tools to support both analysis and simulation of alternative approaches to sampled monitoring. These programs are given in the appendices. Graduate student research support provided during this contract has resulted in an MS thesis.

Table 5.1 Results Fall 1978

1. Gault and Ardalan, "Referenceless On-line Monitoring", Proceedings of the Government Microcircuits Applications Conference, Monterey, CA, November 1978.
2. _____, "Sequential Circuit Output Probabilities From Network Realizations", Submitted to the 1979 International Fault Tolerant Computing Symposium to be held in June 1979.
3. _____, "A Statistical Pin Fault Model", Submitted to The Johns Hopkins Conference on Information Sciences and Systems to be held in March 1979.
4. "Statistical Fault Monitoring of Sequential Circuits" for Mr. Sasan H. Ardalan. This thesis is in the final stages of preparation and will be included in the next report.

5.2 Future Work

The results in statistical fault monitoring [Table 5.1-1,2,3] have illuminated the next level of questions to be addressed. The direction of proposed work can be categorized into three types of effort:

- 1) Continued development of theoretical concepts and analysis models for sampled monitoring [Section 5.2.1],
- 2) Specification and development of software tools for analysis and simulation modeling in support of the theoretical concept development [Section 5.2.2], and

- 3) Experimental validation and "proof of concept" of both theoretical results and simulation models utilizing a hardware textbook [Section 5.2.3].

5.2.1 Theoretical Concept Development

Two fundamental issues which have surfaced in the development of the theoretical approaches to generating network probability responses are the ideas of: 1) signal statistic independence in feedback environments and 2) signal statistic stationarity in operational system environments. Issue 1 can be attached as a theoretical question and preliminary results are given in [Table 5.1 (2)]. Initially, issue 2 can best be approached experimentally, (see Section 5.2.3) and then treated theoretically. The crux of the theoretical development concerning both independence and stationarity is to be able to derive a monitoring strategy and to quantify its sensitivity to these properties.

We have noted that one important measure of effectiveness of a statistical fault monitor is the probability of escape P_{esc} . P_{esc} is a function of the length of test (N), the range of values of input statistics, and the output probed.

For a fixed output probe and a fixed value of N , suppose we plot P_{esc} as a function of the input statistics. If there are many input lines, the plot will be multidimensional. It is reasonable to assume a threshold on P_{esc} (say P_t) so that if actual value of $P_{esc} > P_t$, we say that the test is unsatisfactory; otherwise it is satisfactory. Clearly "valleys" in the above plot are desirable. From the plot (or analytically) we can predetermine the ranges of input statistics for which the test is satisfactory.

If we were conducting the test off-line, then it is clear that we will control the input statistics to be in the desirable range determined above. In the on-line environment, however, we are not able to control the input statistics. But we can probe all input lines and make on-line estimates of the corresponding statistic. Now if the estimated input statistics fall in the desirable range, then we are tempted to conclude that the test is satisfactory. However, if the input statistic is non-stationary, then the above conclusion may be false.

Although, input statistics are likely to be non-stationary in general, we surmise that stationarity exists in a "local" sense. The behavior of inputs

may imply a sudden change in input probabilities. This phase-type behavior is analogous to program behavior in a paging system. Indeed if we probe the page indices (which are the inputs to the memory system) they are likely to exhibit such a behavior. If we assume such a phase-type behavior, then the estimates of input statistics as will be reliable enough for us to make conclusions about the test. Of course, the length N of the test should be short enough to avoid embracing several phases and long enough to provide a good estimate of the input statistic (Ct. law of large numbers). Additional factors affecting the choice of N are the MTDF (which is likely to increase with N), the P_{esc} (which is likely to decrease with N), and P_{FA} (which decreases with an increase in N). Determining the optimal value of N , is thus a very interesting problem that we hope to solve.

The next question is what do we do when it is determined that the input statistics outside the desirable range and hence the test is not satisfactory. There appear to be several alternatives: a) probe another output point. In other words, assume that for each range of input statistics, we have predetermined that most desirable output probe. b) change the length of test N . Thus we will be working with a dynamically changing test length. c) declare the inability of the on-line monitor to perform a satisfactory test. We may take the unit off-line for further tests.

The former two alternatives imply quite a few interesting problems, both, at the theoretical and experimental level.

In addition to these concerns, there are other issues of theoretical concern (given in Table 5.2) which also require concentrated effort.

Table 5.2 Theoretical Issues of Importance to Sampled Monitoring

1. Signal statistic independence in feedback environments.
2. Signal statistic stationarity in operational environments.
3. Statistical fault models which are accurate and computationally feasible.
4. Selection of monitoring points in a network for increased fault detection potential.
5. Investigation of the effectiveness of a variety of monitoring schemes including statistics other than the probability of an output being one.
6. Sequential ratio testing - Variable length monitoring.

5.2.2 Software Analyses and Simulation Tools

In order to employ the techniques being developed to provide built-in-testing capability, a designer would like to be able to:

- 1) Determine a set of suitable monitoring points for a module,
- 2) Determine the expected performance at these monitor points a priori to on-line operation, so that the monitoring device may be parameterized,
- 3) Determine performance trade-off which results from various choices of monitoring points, experiment length, and performance stringency. The primary performance characteristics are:
 - a) probability of escape,
 - b) probability of false alarm, and
 - c) detection latency
- 4) Simulate the performance obtained for a given set of parameterized monitors and assumed fault population.

Preliminary effort has been undertaken in some of these areas. However, there is a need for overall specification and planning to allow for reasonable long-term progress.

In the process of doing research, offhanded tool development is often undertaken. However, in the long run, serious effort directed explicitly toward tool specification and development is required to obtain a meaningful capability.

5.2.3 Experimental Work

Experimental work has a solid reputation in certain areas of study and so precedent does exist for experimental research. The area of fault-tolerant computing is reasonable for experimental projects. The credibility of theoretical claims which are otherwise unvalidated in a practical way can be validated experimentally. The experimental work proposed here has begun as a correlary effort to the more central theoretical effort. A PDP-8 of the late 1960's provides a vehicle by which actual test point monitoring and fault insertion can be employed since the technology of the machine is SSI (see Figure 5.1).

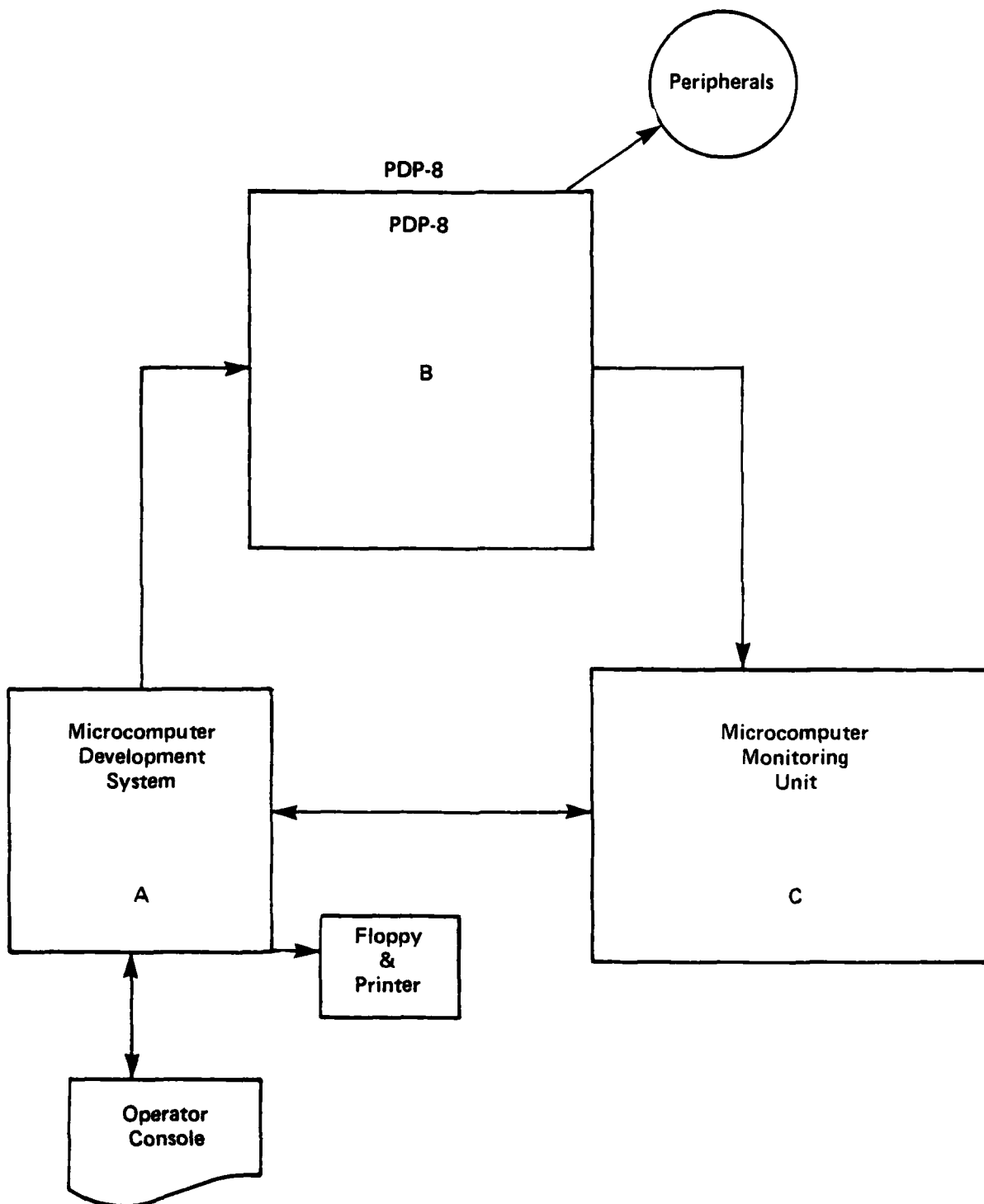


Figure 5.1. Experimental Environment.

The primary goal of this effort is to carry out hard experiments which are strongly related to soft experiments done by means of simulation (Section 5.2.2). This will include the important signal statistic stationarity while executing operational programs (Section 5.2 introduction).

The Microcomputer Development System (labeled A in Figure 5.1) will be used as the primary user interface. This module will provide the capabilities defined in Table 5.3. After an experiment is defined, the target machine (labeled B) will begin executing an operational program. Unit C will sample, according to parameterization from A, the activity of the execution. Results of sampling will be reported back to unit B from C and will be logged along with the experimental conditions.

Table 5.3 Statistical Experimentation Functional Capabilities

Unit A.	Operator Input and Experiment Control/Logging
1.	Specify fault model including insertion points and duration.
2.	Experiment supervision including length, stringency, and other parameters passed to unit C.
3.	File capability for experiments and results.
Unit B.	Target System
1.	Run operational programs.
2.	Control program selection and repetition.
3.	Run self-diagnosis programs.
Unit C.	Monitor Module
1.	Sample operational activity of unit B.
2.	Accepts parameters [length and frequency of samples, stringency, monitor points].
3.	Report results to Unit A.

The experimental work proposed seems to be a very natural and essential next step in developing the ideas of operational fault monitors. An obvious application for such monitors is to embed them within VHSI/VLSI chips in order to report back device status and thus reduce error latency and data pollution. Experimenting in an SSI environment is an attractive first step prior to tackling the more sophisticated LSI world. The outcome of this work will be to transition important theoretical ideas into an important application.

PRECEDING PAGE BLANK-NOT FILLED

APPENDIX A
PROGRAMS FOR COMPUTATION AND SIMULATION

PRECEDING PAGE BLANK-NOT FILMED

```
// EXEC FTGGG
//C.SYSIN DD *
C PROGRAM NAME PSCAN
C *****
C *
C * THIS PROGRAM COMPUTES THE FAULT DENSITY FUNCTION OF AN N-BIT PROGR-
C * AMMABLE COUNTER. THE PROBABILITY OF ESCAPE IS ALSO COMPUTED FOR A
C * SPECIFIED TEST LENGTH AND STRINGENCY.
C * THE PROGRAM IS EASILY MODIFIED SUCH THAT ANY SEQUENTIAL CIRCUIT CAN
C * BE ANALYSED. ONLY THE FUNCTION RELATING OUTPUT TO INPUT STATISTICS *
C * NEED BE SPECIFIED. IN THE PROGRAM:
C * AIN IS AN ARRAY CONTAINING THE INPUT STATISTICS.
C * NP IS THE NUMBER OF PINS.
C * NBIT IS THE NUMBER OF BITS( SAY, # OF PRESET INPUTS).
C * ALPHA IS THE EXPONENT CCEFF. OF THE PIN FAULT PROBABILITY FUNCTION
C * G(K) FOR NP PINS.
C * CNSTAT IS A ROUTINE THAT COMPUTES THE STATE STATISTICS GIVEN INPUT
C * STATISTICS FOR AN N-BIT PROG. COUNTER.
C * IX IS THE TEST LENGTH AND DELTA IS THE STRINGENCY.
C * FOR A FAULTY CONFIGURATION CORRESPONDING TO 0<K<=3**NP-1, G(K) IS
C * THE PROBABILITY OF THAT FAULTY CONFIGURATION OCCURING.
C * ZF(K) IS THE OUTPUT STATISTIC CORRESPONDING TO FAULTY CONFIGURATION
C * K. PHI(Z) IS THE ARRAY CONTAINING THE FAULT DENSITY FUNCTION.
C * PESC IS A ROUTINE THAT COMPUTES THE ESCAPE PROBABILITY GIVEN THE
C * FAULT DENSITY FUNCTION, THE TEST LENGTH, TEST STRINGENCY AND FAULT-
C * FREE OUTPUT STATISTIC.
C * SASAN ARDALAN, SUMMER, 1978.
C *****
0001 DIMENSION FFREE(256)
0002 DIMENSION PIK(21,101)
0003 DIMENSION INF(8,35,8),LN(8)
0004 DIMENSION RC(16),AIN(256),S(256),PC(256)
0005 DIMENSION AIN2(8),H(8)
0006 DIMENSION PHI(1001),F(1001)
0007 DIMENSION S(3000),ZF(2040)
0008 INTEGER A(8)
0009 REAL L
0010 DATA AIN/0.9,0.4,0.9,0.1,0.3,0.3,0.6/
0011 DATA FFREE/0.22875,0.20039,0.12028,0.07744,0.084315,0.0715,
0012 00.05,0.0395,0.02699,0.0223,0.0135,0.0104,0.00934,0.00904,
0013 00.0055,0.0043/
C
C DATA FOR CALCULATION OF PHI(Z)
C
0012 AW3=2.0/0.1040893
0013 ALPHA=ALCGIARG1
C
C NPX1 IS THE NUMBER OF INCREMENTS FOR THE INPUT STATISTIC L.
C DX IS THE INCREMENT VALUE.
C NPZ IS THE NUMBER OF ELEMENTS IN PHI(Z)
C
0014 NPZ=1000
0015 NPX1=11
0016 DX=1.0/(NPX1-1)
C
C NP=NUMBER OF PINS
C
```



```

0017      NP=7
0018      NITE=4
0019      N=1
0020      DO 7 I=1,NBIT
0021      2 NFN=2
0022      DO 1050 ISTATE=1,1

C
C      VARY INPUT STATISTIC CORRESPONDING TO JIN(JL) WHILE KEEPING
C      REMAINING INPUT STATISTIC FIXED. ALWAYS THE PROB. OF ESCAPE
C      BE OBTAINED AS A FUNCTION OF AN INPUT STATISTIC.
C
0023      DO 1040 JL=1,4
0024      TEMP=AIN(JL)
0025      DO 1020 IG=1,NP*1
0026      AIN(JL)=(IG-1)*DX
0027      F=AIN(1)
0028      L=AIN(2)
0029      PT=AIN(3)
0030      DO 371 MI=1,NBIT
0031      571 AIN2(MI)=AIN(3+MI)

C
C      COMPUTE FAULT-FREE STATISTICS.
C
0032      CALL CNSTAT(F,L,PT,FFREE,IBIT,PC,BC,PJST,AIN2,N)
0033      ZJ=FFREE(ISTATE)
0034      WRITE(3,776)
0035      776 FORMAT(//10X,'*** INPUT PROBABILITY VECTOR ***')
0036      WRITE(3,777)(AIN(NK),NK=1,NP)
0037      777 FORMAT(10X,F8.5/)
0038      DO 5 K=1,NP
0039      5 LN(K)=0
0040      NF=1
0041      DO 6 J=1,NP
0042      6 NF=NF*2
0043      NF=NF-1
0044      DO 10 I=1,NF
0045      CALL RIGOFF(I,NP,BC)
0046      NB=0
0047      DO 15 K=1,NP
0048      IF(BC(K).EQ.0)NB=NB+1
0049      15 CONTINUE
0050      LN(NB)=LN(NB)+1
0051      DO 20 K=1,NP
0052      20 INF(NP,LN(NB),K)=BC(K)
0053      10 CONTINUE

C
0054      COMPUTE G(K), ZF(K).

C
C      KK=0
C
C      "I" IS THE NUMBER OF PINS AT FAULT.
C
0056      DO 40 I=1,NP
0057      KI=1

```

```

0058      PR=EXP(-ALPHA*Z)
0059      KK=1
0060      DO 41 K=1,K1
0061  41  NK=NK+2
0062      DO 45 K=1,NK
0063      CALL HICOFF(K,K1,HC)
0064      LNN=LN(1)
0065      DO 46 J=1,LNN
0066      DO 48 IK=1,NP
0067  43  A(IK)=2
0068      IL=0
0069      DO 47 IK=1,NP
0070      IF(INF(1,J,IK).NE.0)GOTO 47
0071      IL=IL+1
0072      A(IK)=BC(IL)
0073  47  CONTINUE
0074      DO 500 MI=1,NP
0075      AR=A(MI)
0076      IF(AIN(MI).NE.0.0)GOTO 499
0077      R(MI)=0.0
0078      IF(AR.EQ.1)B(MI)=1.0
0079      GOTO 500
0080  499  B(MI)=AR/2.0*(1.0+(AR-2.0)*(AIN(MI)-2.0)/A(MI)+AIN(MI)
0081  500  CONTINUE
0082      R=B(1)
0083      L=B(2)
0084      DT=B(3)
0085      DO 501 MI=1,NPIT
0086  501  AIN2(MI)=R(3+MI)
C
C
0087      CALL CNSTAT(R,L,PT,3,NBIT,PC,DC,ZEST,AIN2,N)
0088      KK=KK+1
0089      G(KK)=PRB
0090      ZF(KK)=S(ISTATE)
0091  46  CONTINUE
0092  45  CONTINUE
0093  40  CONTINUE
C
C CALCULATION OF PHI(Z)
C
0094      DZ=1.0/NPZ
0095      DZ2=DZ/2.0
0096      NPZ1=NPZ+1
0097      NP(1,1,1)=ISTATE
0098  113  FORMAT(10X,'STATE NUMBER:',1,1,1)
0099      DO 600 I=1,NPZ1
0100      Z=(I-1)*DZ
0101      ZL=Z-DZ2
0102      ZH=Z+DZ2
0103      SUM=0.0
0104      DO 650 K=1,KK
0105      IF((ZF(K).GT.ZL).AND.(ZF(K).LT.ZH))SUM=SUM+G(K)
0106  650  CONTINUE
0107      PHI(I)=SUM
0108      F(I)=2

```

```

0109      GO TO 1000
C
C      CALCULATE PROBABILITY OF ESCAPE
C
0110      IX=10000
0111      DELTA=0.01
0112      CALL RESC(PHI,ZJ,DELTA,NEZI,PROB,IX)
0113      WRITE(3,116)ZJ,DELTA,IX,PROB
0114      116 FORMAT(//10X,'ZJ= ',F8.5,' WINDOW=',F7.4,' % OF APPLIED IMP
          *VECTORS=',F8.7//10X,'PROBABILITY OF ESCAPE:',E14.7)
C
C      END OF CALCULATION OF PROB. ESCAPE
C
0115      1020 CONTINUE
0116      AIN(JL)=TEMP
0117      1030 CONTINUE
0118      1050 CONTINUE
0119      STOP
0120      END

```

```

0001      SUMMATION OF STATE STATISTICS.
C *****
C * THIS ROUTINE COMPUTES THE STATE STATISTICS OF AN N-BIT PROGRAMMABLE
C * COUNTER SIMILAR TO THE 74163 GIVEN THE INPUT STATISTICS.
C * R,L,AND PT ARE THE RESET,LOAD,AND COUNT ENABLE INPUT STATISTICS.
C * A IS THE ARRAY OF INPUT PRESET STATISTICS LSB FIRST.
C * S IS AN ARRAY CONTAINING THE COMPUTED STATE STATISTICS.
C * NB IS THE NUMBER OF BITS.
C * N=2**NB IS AN INPUT.
C * PC, BC ARE WORKING ARRAYS.
C * RES1 IS THE STATISTIC OF THE STATE ZERO AND IS AN OUTPUT.
C * BICCOFF IS A ROUTINE THAT GENERATES THE BINARY COEFFICIENTS OF A
C * GIVEN DECIMAL INTEGER.
C *
C * SASAN ARDALAN, SUMMER, 1979.
C *****
0002      DIMENSION A(NB),S(4),PC(N),BC(10)
0003      REAL L,KP
0004      ALFA=1.0-(1.0-PT)*LEN
0005      BETA=R*PT*L
0006      GAMMA=R*(1.0-L)
C      CALCULATION OF PC(K)
C
0007      DO 1 I=1,N
0008      CALL BICCOFF(1,NB,HC)
0009      KP=1.0
0010      DO 2 K=1,NB
0011      2 KP=((1.0-A(K))*(-1.0+2.0*A(K))-PC(K))*KP
0012      PC(I)=KP
0013      1 CONTINUE
C      CALCULATION OF 1.0-GAMMA*BIGGERS(BETA/ALFA)*(K/1.0+BIG(BETA/ALFA))
C
0014      SM1=0.0
0015      SM3=0.0
0016      BETAL=BETA/ALFA
0017      PDA=1.0
0018      DO 10 I=2,N
0019      PDA=PDA*BETAL
0020      SM1=SM1+PDA
0021      II=N-I+1
0022      PLA=1.0
0023      DO 25 J=1,II
0024      25 PLA=PLA*BETAL
0025      SM3=SM3+PC(I)*(1.0-PLA)
0026      10 CONTINUE
0027      SM3=SM3/(1.0-BETAL)
0028      RES1=((1.0-GAMMA*SM3/ALFA)/(1.0+SM1))
0029      DO 30 J=2,N
0030      HAJ=1.0
0031      DO 33 LL=2,J
0032      33 HAJ=HAJ*BETAL
0033      SM1=0.0
0034      IF(BETAL.EQ.0)GOTO 37
0035      DO 30 K=2,J
0036      KI=J-K+1
0037      HCAK=1.0

```

0038	DO 31 M=1,N1
0039	31 BDAK=BDAK/HETAL
0040	BDAK=BDAK/HETAL
0041	30 SM1=SM1+BDAK*PC(K)
0042	32 RES2=SM1*GAMMA/ALFA
0043	S(J)=RAJ*RES1+RES2
0044	50 CONTINUE
0045	S(1)=RES1
0046	RETURN
0047	END

```

0001      ***** BINARY COEFFICIENTS *****
C      *
C      * THIS ROUTINE GENERATES THE BINARY COEFFICIENTS OF A DECIMAL INTEGER *
C      * I. I IS THE DECIMAL INTEGER INPUT. *
C      * BC IS AN ARRAY CONTAINING BINARY COEFFICIENTS( MSB FIRST). *
C      *  $I = BC(1) + 2 * BC(2) + 4 * BC(3) + 8 * BC(4) + \dots + 2^{(NB-1)} * BC(NB)$ . *
C      *
C      *****
0002      DIMENSION BC(1)
0003      M=1
0004      N=1.0
0005      DO 10 K=1,NB
0006      10  N=N*2.0
0007      DO 20 K=1,NB
0008      N=N/2.0
0009      IF (M.LT.N) GO TO 1
0010      BC(K)=1.0
0011      M=M-N
0012      GO TO 20
0013      1  BC(K)=0.0
0014      20 CONTINUE
0015      RETURN
0016      END

```

```

0001      SUBROUTINE PRCB(N, ZJ, DELTA, PRGB, PRCH)
C      *
C      * THE FOLLOWING SUBROUTINE CALCULATES THE PROBABILITY OF ESCAPE.
C      * PHI IS AN ARRAY WITH N ELEMENTS CONTAINING THE FAULT DENSITY
C      * FUNCTION.
C      * ZJ IS THE FAULT-FREE OUTPUT STATISTIC.
C      * IX IS THE TEST LENGTH.
C      * DELTA IS THE TEST STRINGENCY.
C      * PRGB IS THE ESCAPE PROBABILITY.
C      *
C      * SASAN ARDALAN, SPRING 1973
C      *
C      *
C      *
0002      DIMENSION PHI(N)
0003      NI=N+1
0004      ZINC=1.0/N
0005      PRGB=0.0
0006      IK=IX*(ZJ-DELTA)
0007      LK=IX*(ZJ+DELTA)
0008      TPX=2.0*3.1415926*IX
0009      IF((IK.LT.0).OR.(LK.GT.N))RETURN
0010      DO 30 I=IK,LK
0011          RK=I
0012          PSUM=0.0
0013          DO 35 K=2,N
0014              IF(PHI(K).EQ.0.0)GO TO 35
0015              FK=K-1
0016              FN=N
0017              ZZ=FK/FN
0018              ZLM=ZZ*(1.0-ZZ)
0019              TP=TPX*ZLM
0020              TEX=(RK-IX*ZZ)*(RK-IX*ZZ)/2.0/IX/ZLM
0021              IF(TEX.GT.100.0)GO TO 35
0022              TEX=-TEX
0023              TEX=EXP(TEX)/SQRT(TP)
0024              PSUM=PSUM+PHI(K)*TEX
0025      35 CONTINUE
0026      PRCH=PRGB+PSUM*ZINC
0027      30 CONTINUE
0028      RETURN
0029      END

```

```

C      PROGRAM NAME CNTRSIM
C      *****
C      *
C      * THIS PROGRAM SIMULATES AN N-BIT PROGRAMMABLE COUNTER SIMILAR TO THE
C      * 74163. THE INPUTS TO THE PROGRAM ARE AS FOLLOWS:
C      * CARD#1: INPUT STATISTICS NR(* OF BITS),R(RESET),L(LOAD),PT(COUNT EN-
C      * ABLE) FORMAT 12,F8.5,2F10.7
C      * CARD#2: PARALLEL LOAD INPUT STATISTICS CONTAINED IN ARRAY A. LSB
C      * ENTERED FIRST. FORMAT 8F10.7
C      * CARD#3: TEST LENGTH( NUMBER OF SIMULATIONS),NSIM. FORMAT I10
C      * CARD#4: STRINGENCY WINDOW,DELTA. FORMAT F10.7
C      * CARD#5: FAULT CONFIGURATION FOR PINS R,L,PT,LSP,...,MSB.
C      * *2* FAULT FREE, *1* STUCK-AT-ONE, *0* STUCK-AT-ZERO. FORMAT 9011.
C      *
C      *
C      * PROGRAM TERMINATES IF 5 DETECTED IN FIRST COLUMN.
C      * OUTPUT CONSISTS OF RESULT OF TEST( PASS/FAIL), SIMULATED AVERAGES,
C      * FAULT FREE STATISTIC FOR EACH STATE.
C      *
C      * CNSTAT: ROUTINE THAT COMPUTES STATE STATISTICS GIVEN INPUT STATISTIC
C      * CNTR: ROUTINE THAT SIMULATES COUNTER GIVEN INPUT STATISTICS AND
C      * NUMBER OF SIMULATIONS. OUTPUTS SIMULATED STATE AVERAGES.
C      * BICCF: ROUTINE REQUIRED BY CNSTAT.
C      * RAND: ROUTINE REQUIRED BY CNTR.
C      *
C      * SASAN ARDALAN, SUMMER, 1978.
C      *
C      *****
0001      DIMENSION A(256),S(256),IA(8),IAC(8),FFREE(256)
0002      DIMENSION T(16),B(16)
0003      DIMENSION PC(16),HC(8)
0004      INTEGER F(16)
0005      REAL L
0006      DATA IAS/97335,11917,66260,76305,80441,83764,33475,21743/
0007      DATA FFREE/0.22873,0.20039,0.12021,0.0714,0.03431,0.015,
      50.05,0.0395,0.02699,0.0228,0.0135,0.0114,0.00934,0.00704,
      50.0055,0.0043/
0008      READ(1,301)NB,R,L,PT
0009      READ(1,302)(A(K),K=1,NB)
0010      300 FORMAT(I10)
0011      301 FORMAT(12,F8.5,2F10.7)
0012      302 FORMAT(8F10.7)
0013      WRITE(3,320)
0014      370 FORMAT(/10X,'NR,R,L,PT: /')
0015      WRITE(3,301)NB,R,L,PT
0016      WRITE(3,327)
0017      322 FORMAT(/10X,'      A,B,C,... /')
0018      WRITE(3,302)(A(K),K=1,NB)
0019      N=1
0020      DO 19 JJ=1,NB
0021      19 N=N*2
C
C      COMPUTE FAULT FREE STATISTICS.
C
0022      CALL CNSTAT(R,L,PT,FFREE,NB,PC,HC,RESI,A,N)
0023      T(1)=R

```



```

0024      T(1)=L
0025      T(3)=PT
0026      DO 520 KK=1,NH
0027      T(3+KK)=A(KK)
0029      520 CONTINUE
0029      NP=NH+3
0030      READ(1,300)NSIM
0031      READ(1,350)DELTA
0032      WRITE(3,310)NSIM,DELTA
0033      310 FORMAT(1H1,10X,' NUMBER OF SIMULATIONS:',110,' XINDEL:',110)
0034      85 CONTINUE
0035      WRITE(3,330)
0036      330 FORMAT(//2(30(' '),//))
0037      READ(1,360)(F(K),K=1,NP)
0038      IF(F(1).GT.2)STOP
0039      350 FORMAT(F10.7)
0040      360 FORMAT(80I1)
0041      DO 500 MI=1,NP
0042      AP=F(41)
0043      IF(T(MI).NE.0.0)GOTO 499
0044      R(MI)=0.0
0045      IF(F(MI).EQ.1)B(MI)=1.0
0046      GOTO 500
0047      499 R(MI)=4F/2.0*(1.0+(AP-2.0)/(T(MI)-2.0)/(41)-1.0)
0048      500 CONTINUE
0049      DO 501 MI=1,NB
0050      A(MI)=R(3+MI)
0051      R=B(1)
0052      L=B(2)
0053      PT=B(3)
C
C      SIMULATE FAULTY CONFIGURATION.
C
0054      CALL CNTRS(R,L,PT,A,S,NH,IA,IAG,(317)
0055      NS=1
0056      DO 49 I=1,NB
0057      49 NS=NS*2
C      DO 50 I=1,NS
0058      DO 50 I=1,J
C
C      PERFORM TEST ON SIMULATED AVERAGES.
C
0059      ZL=FF4FE(I)-DELTA
0060      ZH=FFREE(I)+DELTA
0061      IF((S(I).GE.ZL).AND.(S(I).LE.ZH))GOTO 61
C
C      PRINT TEST RESULTS FOR EACH STATE.
C
0063      WRITE(3,370)(F(K),K=1,NP)
0064      370 FORMAT(//10X,'FAULTY CONFIGURATION',1711,' FAILED THE TEST
0065      GO TO 50
0066      51 WRITE(3,372)(F(K),K=1,NP)
0067      372 FORMAT(//10X,'FAULTY CONFIGURATION',1711,' PASSED THE TEST
0068      50 WRITE(3,375)1,S(1),FFEE(1)
0069      375 FORMAT(//10X,'STATE',15,' STATE PROB.',F10.7,' FAULT FREE
1,F10.7,//30X,70(' '),//)
0070      GOTO 85
0071      STOP
0072      5      END

```

```

0001      SUBROUTINE RANDU(P,IX)
C .....
C *
C * THIS ROUTINE GENERATES A RANDOM ONE WITH PROBABILITY P.
C * J IS THE RANDOM ONE OUTPUT.
C *
C .....
0002      CALL RANDU(IX,IY,YFL)
0003      J=I
0004      IF(YFL.GT.P)J=0
0005      IX=IY
0006      RETURN
0007      END

```

```

0001      *****
C      *
C      * THIS ROUTINE SIMULATES AN 1-BIT PROGRAMMABLE COUNTER SIMILAR TO THE
C      * 74163. R,L,PT ARE THE RESET,LOAD, AND COUNT ENABLE INPUT STATISTICS.
C      * A IS THE ARRAY CONTAINING THE PRESET INPUT STATISTICS LSB FIRST.
C      * NB IS THE NUMBER OF BITS( PRESET INPUTS).
C      * S IS THE ARRAY OF SIMULATED STATE AVERAGES.
C      * IAS IS AN ARRAY CONTAINING NB RANDOM INTEGERS(3-5 DIGITS).
C      * IA IS A WORKING ARRAY.
C      * NSIM IS THE NUMBER OF SIMULATIONS.
C      * RAND IS A ROUTINE THAT GENERATES A RANDOM ONE WITH SPECIFIED PROB-
C      * ABILITY.      SASAN ARDALAN, SUMMER, 1978.
C      *
C      *****
0002      DIMENSION A(1),S(1),IA(1),IAS(1)
0003      REAL L
0004      DATA IFR,IL,IPT/38329,80973,01131/
0005      N=1
0006      DO 5 I=1,NB
0007      S(N)=2
0008      DO 15 J=1,N
0009      15 S(J)=0.0
0010      J=1
0011      DO 55 KK=1,NSIM
C
C      GENERATE RANDOM INPUTS
C
0012      CALL RAND(R,IF,IFR)
0013      CALL RAND(L,IL,IL)
0014      CALL RAND(PT,IPT,IPT)
0015      DO 12 K=1,NB
0016      SA=A(K)
0017      IXA=IAS(K)
0018      CALL RAND(SA,IAG,IXA)
0019      IA(K)=IAG
0020      IAS(K)=IXA
0021      12 CONTINUE
0022      IF(IF.EQ.0)GOTO 1
0023      IF(IL.EQ.0)GOTO 2
0024      IF(IPT.EQ.0)GOTO 50
0025      J=J+1
0026      IF(J.EQ.N+1)J=1
0027      GOTO 50
0028      1 J=1
0029      GOTO 50
0030      2 ISUM=0
0031      KE=1.0
0032      DO 20 K=1,NB
0033      ISUM=KE*IA(NB-K+1)+ISUM
0034      KE=KE*2
0035      20 CONTINUE
0036      J=(ISUM+1
0037      50 S(J)=S(J)+1.0
0038      55 CONTINUE
0039      DO 25 J=1,N

```

```

0040      25 S(J)=S(J)/NSIM
C        WRITE(3,325)
C        WRITE(3,100)(S(J),J=1,N)
0041      325 FORMAT(1H1,/,10X,' SIMULATED RESULTS')
0042      100 FORMAT(/,10X,E14.7)
0043      RETURN
0044      END

```

EXAMPLE DATA FOR CNTRSIM

```

//G.SYSIN CD *
  4 0.9      0.4      0.9
  0.9      0.3      0.3      0.5
    10000
  0.01
2222222
1222222
2022222
2220222
2202222
2222202
2222220
2122222
2212222
2221222
2222122
2222212
2222221
1212222
1012222
2020002
2220001
022222
222022
555555
    1000
  4 .9      0.4      0.9
  0.1      0.3      0.3      0.5
  0.01
    1000000
  4 .9      0.4      0.9
  0.1      0.3      0.3      0.5
/*

```

```

C      PROGRAM NAME SHIFTSIM
C      *****
C      *
C      * THIS PROGRAM SIMULATES THE STATISTICAL TESTING OF A TWO BIT PARALLEL
C      * LOAD, SHIFT RIGHT, SHIFT REGISTER. INPUTS TO THE PROGRAM ARE:
C      * CARD#1: INPUT STATISTICS S,M,A,B  FORMAT 8F10.7
C      * CARD#2: NUMBER OF SIMULATIONS( TEST LENGTH)  FORMAT I10
C      * CARD#3: STRINGENCY WINDOW  FORMAT F10.7
C      * CARD#4: FAULT CONFIGURATION FOR PINS S,M,A,B. THE CONFIGURATION IS
C      * SPECIFIED BY '2' FAULT FREE, '1' STUCK-AT-ONE, '0' STUCK-AT-ZERO.
C      * THE FORMAT IS 8011. EACH FAULT CONFIGURATION IS SPECIFIED ON A
C      * SEPARATE CARD.
C      * THE PROGRAM TERMINATES WHEN A 5 IS PUNCHED IN COLUMN 1.
C      * PRINTOUT IS CONSISTED OF WHETHER THE CIRCUIT PASSES OR FAILS THE
C      * TEST, THE FAULT CONFIGURATION, THE FAULT FREE STATISTIC, AND THE
C      * SIMULATED AVERAGE.
C      *
C      * SHIFTS: SUBROUTINE THAT SIMULATES SHIFT REGISTER
C      * SHIFT: SUBROUTINE THAT COMPUTES STATE PROBABILITIES OF SHIFT REG.
C      * OTHER ROUTINE REQUIRED RAND.
C      * NP=NUMBER OF PINS
C      * NB= NUMBER OF BITS
C      * DELTA= TEST STRINGENCY WINDOW.
C      * F(K): ARRAY CONTAINING FAULT CONFIGURATION.
C      * T(K): ARRAY CONTAINING INPUT STATISTICS.
C      *
C      *****
0001      DIMENSION PP(4),PT(1000)
0002      DIMENSION T(4),S(4),FFREE(4)
0003      DIMENSION B(4)
0004      INTEGER SS(4,2),F(4)
0005      NP=4
0006      NB=2
0007      READ(1,307)(T(I),I=1,4)
0008      307 FORMAT(8F10.7)
0009      READ(1,300)NSIM
0010      300 FORMAT(I10)
0011      READ(1,350)DELTA
0012      WRITE(3,310)NSIM,DELTA
0013      310 FORMAT(1H1,10X,' NUMBER OF SIMULATIONS:',I10,' WINDOW:',F10.7)
0014      WRITE(3,333)(T(I),I=1,4)
0015      333 FORMAT(//10X,' INPUT PROBABILITY VECTOR S,M,A,B. '//10X,4(F10.7))
C
C      SIMULATE FAULT FREE CIRCUIT.
C
0016      CALL SHIFTS(T,S,NSIM,SS,PP,PT)
0017      CALL SHIFT(PP,FFREE)
0018      85 CONTINUE
0019      WRITE(3,330)
0020      330 FORMAT(//2(30(' **')//1)
0021      READ(1,360)(F(K),K=1,NP)
0022      IF(F(1).GT.2)STOP
0023      350 FORMAT(F10.7)
0024      360 FORMAT(8011)
0025      DO 300 M=1,NP
0026      AC=F(M)

```

```

0027      I=(T(MI).NO.0.0)GO TO 49
0028      R(MI)=0.0
0029      IF (MI.EQ.1)R(MI)=1.0
0030      GOTO 500
0031      499 F(MI)=AP/2.0*(1.0+(AB-2.0)*T(MI)-2.0/7*(MI)*T(MI))
0032      500 CONTINUE
C
0033      C      SIMULATE FAULTY CONFIGURATION.

C
0034      CALL SHIFTS(B,S,NSIM,SS,PP,PT)
0035      NS=1
0036      DO 49 I=1,NB
0037      49 NS=NS*2
0038      DO 50 I=1,NS
C
C      CHECK IF SIMULATED OUTPUT WITHIN TEST WINDOW OF FAULT FREE
C
0039      ZL=FFREE(I)-DELTA
0040      ZH=FFREE(I)+DELTA
0041      IF((G(I).GE.ZL).AND.(S(I).LE.ZH))GOTO 51
C
C      PRINT TEST RESULTS FOR EACH STATE
C
0042      WRITE(3,370)(F(K),K=1,NP)
0043      370 FORMAT(/10X,'FAULTY CONFIGURATION ',4I1,' FAILED THE TEST')
0044      GO TO 50
0045      51 WRITE(3,372)(F(K),K=1,NP)
0046      372 FORMAT(/10X,'FAULTY CONFIGURATION ',4I1,' PASSED THE TEST')
0047      50 WRITE(3,375)I,S(I),FFREE(I)
0048      375 FORMAT(/10X,'STATE ',15,' STATE FREE ',F10.7,' FAULT FREE
        S,F10.7//30X,70('**')/)
0049      WRITE(3,388)PP
0050      388 FORMAT(/10X,'INPUT MEASURES(S,M,A,N):',4(F7.4,3X))
0051      GOTO 35
0052      STOP

0053      END

```

```

0001      SUBROUTINE SHIFTG(PV,NSIM,SS)
C      *****
C      *
C      * THIS ROUTINE SIMULATES A 2-BIT PARALLEL-LOAD, RIGHT-SHIFT,SHIFT
C      * REGISTER. PV IS THE ARRAY CONTAINING INPUT STATISTICS S,M,A, AND B.
C      * S IS AN ARRAY CONTAINING THE RESULTANT SIMULATED AVERAGES FOR EACH
C      * STATE. NSIM IS THE NUMBER OF SIMULATIONS.
C      * SS IS A TWO DIMENSIONAL WORKING ARRAY.
C      * SUBROUTINE RAND GENERATES A RANDOM ONE WITH SPECIFIED PROBABILITY.
C      *
C      * SASAN ARDALAN SUMMER, 1978.
C      *****
0002      DIMENSION PV(4),S(4)
0003      INTEGER SS(4,2),PS,SR,A,B,M
0004      DATA IM,IS,IA,IB/3832,66973,61131,76220/
0005      SS(1,1)=1
0006      SS(2,1)=4
0007      SS(3,1)=4
0008      SS(2,2)=3
0009      SS(1,2)=2
0010      SS(4,1)=1
C
0011      CALL RAND(IM,M,IM)
0012      CALL RAND(IS,SR,IS)
0013      CALL RAND(IA,A,IA)
0014      CALL RAND(IB,B,IB)
0015      IF(M.EQ.0)GOTO 10
0016      PS=A+B*2+1
0017      IF(PS.EQ.3)PS=4
0018      IF((A*B).EQ.1)PS=3
0019      S(PS)=S(PS)+1.0
0020      GOTO 35
0021      10 PS=SS(PS,SR+1)
0022      S(PS)=S(PS)+1.0
0023      35 CONTINUE
0024      DO 14 I=1,4
0025      14 S(I)=S(I)/NSIM
0026      RETURN
0027      END

```

```

0001      SUBROUTINE SHIFT(1,2)
0002      *****
0003      C *
0004      C * THIS SUBROUTINE COMPUTES THE STATE PROBABILITIES OF A 2-BIT PARALLEL-
0005      C * LOAD, RIGHT-SHIFT, SHIFT REGISTER.
0006      C * Q IS THE ARRAY OF COMPUTED STATE STATISTICS.
0007      C *
0008      C *****
0009      DIMENSION T(NP),Q(4)
0010      REAL M,MC
0011      S=T(1)
0012      M=T(2)
0013      A=T(3)
0014      B=T(4)
0015      AC=1.0-A
0016      BC=1.0-B
0017      SC=1.0-S
0018      MC=1.0-M
0019      Q(1)=SC*MC-S*SC*MC*MC-SC*A*M*MC+B*AC*M
0020      Q(2)=S*MC-S*S*MC*MC-S*A*M*MC+B*AC*M
0021      Q(3)=S*S*MC*MC+A*S*MC*M*M*A*B
0022      Q(4)=MC*MC*S*SC+MC*M*SC*A+B*AC*M
0023      RETURN
0024      END

```

EXAMPLE DATA: SHIFTSIM

```

//G.SYSIN DD *
0.6      0.9      0.7      0.7
10000
0.01
2222
2122
1222
2212
2221
0222
2022
2202
2220
2102
2101
0201
2111
2112
3555
/*

```



```

C *****
C *
C * THIS PROGRAM COMPUTES THE DOMINANT COEFFICIENT, ALPHA, OF THE FIRST-
C * FAULT PROBABILITY G(X).
C * NP IS THE NUMBER OF INPUT PINS OF THE PACKAGE(CIRCUIT).
C *
C *****
0001      EXTERNAL FCT,COMB
0002      COMMON NP
0003      IEND=500
0004      EPS=1.0E-4
0005      XLI=0.0
0006      XRI=1.0
0007      DO 20 NP=1,NP
C
C      RTM1 CAN BE FOUND IN THE IBM SCIENTIFIC SUBROUTINE PACKAGE.
C
0008      CALL RTM1(X,F,FCT,XLI,XRI,EPS,IEND,IER)
0009      *WRITE(3,100)X,F,XLI,XRI,EPS,IEND,IER
0010      100  FORMAT(' ',10X,'X,F,XLI,XRI,EPS,IEND,IER',//5(5X,F11.7)//10X,
0011             XX=2.0/X
0012             ALPHA=ALOG(XX)
0013             *WRITE(3,250)ALPHA,NP
0014      250  FORMAT('//10X,'***** ALPHA *****',E11.7,' *** / OF PINS ***',
             $13)
0015      20  CONTINUE
0016      STOP
0017      END

0001      FUNCTION FCT(X)
0002      EXTERNAL COMB
0003      COMMON NP
0004      SUM=0.0
0005      XK=1.0
0006      DO 40 I=1,NP
0007      XK=XK*X
0008      SUM=SUM+COMB(NP,I)*XK
0009      40  CONTINUE
0010      FCT=SUM-1.0
0011      RETURN
0012      END

0001      FUNCTION COMB(N,K)
0002      IF(K.EQ.1)COMB=N
0003      IF(K.EQ.N)COMB=1.0
0004      IF((K.EQ.1).OR.(K.EQ.N))RETURN
0005      X=1.0
0006      NMK=N-K
0007      DO 20 I=1,N
0008      X=X*I
0009      IF(I.EQ.K)XK=X
0010      IF(I.EQ.NMK)XNMK=X
0011      20  CONTINUE
0012      COMB=X/XK/XNMK
0013      RETURN
0014      END

```

APPENDIX B
PROGRAM FOR 3-D PLOTS

PRECEDING PAGE BLANK-NOT FILMED

```
//**F**XXXXX
// EXEC FTGCG
//C.SYSIN CD A
C PROGRAM NAME PLOTCT
C *****
C *
C * THIS PROGRAM PLOTS THE STATE PROBABILITY OF A COUNTER VERSUS THE *
C * VARIATION OF TWO OF ITS INPUT STATISTICS WHILE THE OTHER STATISTICS *
C * REMAIN FIXED. *
C * CNSTAT IS A ROUTINE THAT RELATES STATE STATISTICS TO INPUT STATISTICS *
C * R,L,PT ARE THE RESET, LOAD, AND COUNT ENABLE STATISTICS OF THE COUNTER. *
C * A IS AN ARRAY CONTAINING THE PRESET INPUT STATISTICS. *
C * PIK(J,I) IS AN ARRAY CONTAINING THE Z COORDINATE CORRESPONDING TO *
C * X(J), AND Y(I). *
C * NPX IS THE NUMBER OF POINTS ON THE X AXIS. *
C * NPY IS THE NUMBER OF POINTS ON THE Y AXIS. *
C * RMIN AND RMAX ARE THE MIN. AND MAX. VALUES OF PIK(J,I) THAT IS, Z. *
C * FCWS CORRESPOND TO THE X AXIS AND COLUMNS TO THE Y AXIS. *
C * CPI IS THE NUMBER OF COLUMNS/INCH AND RPI THE NUMBER OF ROWS/INCH. *
C * UNIT 8 IS THE DISC WHERE PIK(J,I) IS STORED FROM WHICH THE PROGRAM *
C * SYMVU OBTAINS THE PLOT INFORMATION. *
C *****
C DIMENSION PIK(50,50),A(256),PC(256),EC(16),S(256)
C REAL L
C READ(1,110)NB
C READ(1,120)(A(K),K=1,NB)
C READ(1,130)R,L,PT
110 FORMAT(12)
120 FORMAT(16F5.3)
130 FORMAT(3F5.3)
C N=1
C DO 5 I=1,NB
5 N=N+2
C READ(1,240)NPX,NPY
240 FORMAT(2I3)
C OX=1.0/NPX
C OY=1.0/NPY
C NPX1=NPX+1
C NPY1=NPY+1
C DO 500 I=1,NPX1
C FT=(I-1)*OX
C DO 500 J=1,NPY1
C LT=(J-1)*OY
C CALL CNSTAT(R,L,PT,S,NB,PC,EC,RES1,A,N)
C PIK(I,J)=S(2)
500 CONTINUE
C NC=NPY1
C NF=NPX1
C RPI=2.0
C CPI=2.0
C RMIN=0.0
C RMAX=0.6
C WRITE(8)NR,NC,RPI,CPI,RMIN,RMAX
C DO 44 J=1,NR
44 WRITE(8)(PIK(J,I),I=1,NC)
C REWIND 8
C CALL EXIT
C END
```

```

//C01001 DD DSN=66RT1.D1 SP=(NEW,PA 00),
// UNIT=DISK,VOL=SER=RT1222,
// SPACE=(TRK,(5,5),RLSE),
// DCE=(RECFM=VBS,LRFL=2060,BLKSIZE=1000,BUFNO=1)
//G.SYSIN DD *
4
0.0 0.3 0.3 0.5
0.9 0.4 0.9
40 40
/*
//S2 EXEC SYMVU
//G.NCSPLCT DD SYSCUT=C
//G.SYSIN DD *
STATS OF CNTR
40 40 2
45.0340.0 4.0 3.0
/*
//G.DATA DD DSN=66RT1.DISP=OLD
/*

```

1 1 1

0.6

APPENDIX C
STATISTICAL FAULT MONITORING
OF SEQUENTIAL CIRCUITS

PRECEDING PAGE BLANK-NOT FILLED

STATISTICAL FAULT MONITORING
OF SEQUENTIAL CIRCUITS

by

SASAN H. ARDALAN

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

DEPARTMENT OF ELECTRICAL ENGINEERING

Raleigh

1 9 7 9

APPROVED BY:

Til Rivin

Nasser Sagan

James H. Gault
Chairman of Advisory Committee

ABSTRACT

ARDALAN, SASAN HOUSTON. Statistical Fault Monitoring of Sequential Circuits. (Under the direction of JAMES W. GAULT.)

The effectiveness of statistical fault monitoring of digital sequential circuits as an on-line built-in-test technique was investigated. In applying this technique it is important to be able to evaluate both the probability of escape and the probability of false alarm. The probability of false alarm has been shown to be a very straightforward function of the experiment length and test stringency. The probability of escape, however, depends on the network fault density function. To derive this function, a statistical fault model was investigated. Also, necessary for the derivation of this function were efficient methods for obtaining output statistics of sequential circuits as a function of input statistics. Various methods were examined and reported for deriving output statistics from the state diagram, regular expression description, and directly from the circuit implementation of sequential circuits. Based on these methods and the fault model considered an algorithm was found for the computation of the network fault density function. Using this function the escape probability for various input statistics and test specifications was derived and the results were confirmed by computer simulations. Test conditions were found where

the fault detection of sample sequential circuits was extremely effective.

Appendices include computer listings of the algorithm and simulation programs. A method for three-dimensional plotting is also included.

BIOGRAPHY

Sasan H. Ardalan was born March 20, 1956, in Tehran, Iran. He received his elementary and secondary education in Tehran, and graduated from Alborz High School in 1974.

After coming to the United States, he began undergraduate study at North Carolina State University where he obtained his Bachelor of Science degree in Electrical Engineering in May 1977 with high honors. During the summer of 1977 he worked part time for University Systems Analysis and Control, the Research Triangle Institute, and the Department of Forestry, North Carolina State University.

The author entered the graduate school of Electrical Engineering at North Carolina State University in August 1977 and worked towards the Master of Science degree.

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude for the advice and guidance given by the Chairman of his Advisory Committee, Professor J.W. Gault, in both the research and writing of this thesis. Also, the author appreciates the help and support of Mr. James Clary of the Research Triangle Institute.

Finally, appreciation is due Ms. Sande Maxim for the time and effort she expended typing and compiling this manuscript.

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
1.1 Motivation.	1
1.2 Test Assumptions and Standard Bit Strategy.	1
1.3 Problem Background and Literature Review.	3
1.4 Research Objectives and Approach.	7
2. STEADY-STATE PROBABILITIES OF SEQUENTIAL CIRCUITS.	9
2.1 Introduction.	9
2.2 Markov Chain Model of Sequential Circuits	9
2.3 Steady-State Probabilities.	12
2.4 Steady-State Output Probabilities from Regular Expressions	22
2.4.1 Introduction	22
2.4.2 Review of regular expressions.	23
2.4.3 Steady-state probabilities	28
2.5 Derivation of Output Probabilities of Iterative Sequential Circuits	30
2.6 State Probabilities Directly from Sequential Circuit Using Regular Expressions	36
2.7 Comparison of Methods	48
3. ANALYSIS OF A FOUR-STATE SEQUENTIAL CIRCUIT.	51
4. STATISTICAL PIN-FAULT MODEL.	64
4.1 Introduction.	64
4.2 Statistical Pin-Fault Model	64
4.3 Derivation of $\phi(z)$	72
4.4 Algorithm for the Calculation of $\phi(z)$	73
5. COMPUTATIONAL RESULTS AND SIMULATION	77
5.1 Introduction.	77
5.2 Escape Probability of a Two-Bit Right-Shift Parallel-Load Shift Register.	77
5.3 Test Simulations for the Shift Register	84
5.4 Probability of Escape of a Four-Bit Synchronous Counter	86
5.5 Computer Simulations.	87
6. CONCLUSIONS AND FURTHER RESEARCH	90

TABLE OF CONTENTS (continued)

	Page
7. LIST OF REFERENCES	92
APPENDICES	94
APPENDIX A Computation and Simulation Programs. . .	95
APPENDIX B Three-Dimensional Plot Routine	113

1 INTRODUCTION

1.1 Motivation

Digital electronic equipment is being widely used in varied environments in the military and industry. Thus digital equipment is used in critical control and computation applications where proper fault-free operation of the equipment is essential. Moreover, most systems utilize modular digital circuits in the construction of their equipment. This has the advantage of having a structure for the equipment and thus providing for ease of maintenance and repair.

Thus, the on-line monitoring of faults while the digital equipment is operating is a basic requirement in large complex modular systems. Moreover, since the trend is toward modular digital equipment, the idea of built-in test has emerged where each module has a built-in circuit that monitors the module for faults. It is the purpose of this study to investigate a standard built-in-test strategy suitable for monitoring faults in a broad range of modules [1].

1.2 Test Assumptions and Standard Bit Strategy

This section provides an outline for the test assumptions and test strategy used for testing on-line digital circuits. The basic test assumptions are [5]:

- The testing objective is detection of faults in modular digital sequential electronic equipment which

has been fielded, i.e., has passed design and manufacturing acceptance tests. The Built-In Test (BIT) circuit will monitor on-line operations of a digital sequential module and will not alter normal processing.

- The testing objective is to detect multiple as well as single logic faults.
- The behavior of a module being monitored will be characterized statistically assuming stationary and independent input statistics.

The referenceless on-line monitoring strategy used to test modules under the assumptions above is summarized below. *A priori*

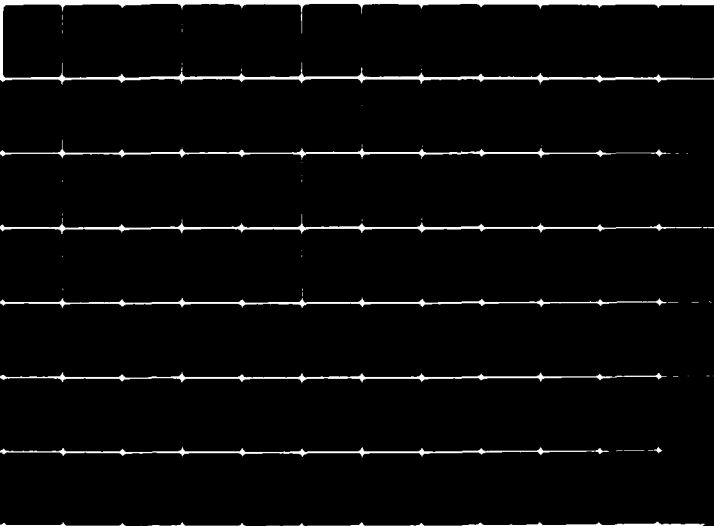
- 1) Define a set of points to be monitored. In general, there will include inputs (x), outputs (z), and internal state values (s).
- 2) Derive, for the states and outputs, an expected value of the statistics $E_s(x)$ and $E_z(x)$ as a function of the input statistic x .
- 3) Derive a test stringency ϵ and test length based on a desired test quality, false alarm rate, and escape rate. This will involve the definition of a fault model and the fault density functions $\phi(x, z)$, $\phi(x, s)$.
- 4) During system execution the fault monitor will
 - For an experiment of length N gather statistics on x , s , and z .

8-0004 000

RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/8 12/1
TECHNIQUES FOR ON-LINE FAULT MONITORING IN MODULAR DIGITAL SYST--ETC(U)
JAN 80 J W SAULT, K S TRIVEDI, P N MARINOS N00039-78-C-0431
RTI/1667/00-01F NL

UNCLASSIFIED

2 of 4
FOIA
014 888



A
84 88

- Indicate a failure if a measured statistic (e.g., m_z) is outside the acceptance window for the expected value of this statistic as a function of the measured input statistic m_x , that is, the experiment fails if $m_z > E_z(m_x) + \epsilon$ or $m_z < E_z(m_x) - \epsilon$ and passes otherwise.

The next section provides a background for the problem, namely analyzing the efficiency of referenceless random testing for BIT, and reviews the current work related to the subject.

1.3 Problem Background and Literature Review

Random test generation methods for digital circuits have been used to overcome some of the computation costs of deterministic methods [8]. In these methods, random inputs are fed into the circuit under test and the outputs are compared with a stored reference or the output generated by a "gold unit". The practicality and effectiveness of these methods have been analyzed in the literature [12,14,16]. Moreover, these studies have been concerned with the error latency of faults in digital combinational circuits. The error latency of a fault set is the number of input vectors applied to a digital circuit while a fault in the fault set is active, until the first incorrect output due to that fault is observed [14]. Most of the studies dealt with combinational circuits; however, the error latency of digital sequential circuits was also analyzed [13]. These studies showed that in many cases random testing involved

much less computation and storage requirements than deterministic methods [14].

Inherent in the study of the random testing of digital circuits is knowledge of formulations for deriving output probabilities as a function of input probabilities. Parker and McCluskey [9] presented an efficient method for relating output to input probabilities for combinational circuits. They also applied these methods to the analyses of logic circuits with faults using input signal probabilities [10]. Relating output to input probabilities, however, is difficult for sequential circuits; and no efficient methods exist. One method is to model the sequential circuit as a Markov chain and derive the steady-state probabilities. A more efficient method is to derive the probabilities from the regular expression description of the sequential circuit [11]. This method, however, has certain shortcomings which limit its application.

Although random testing methods using a reference or "gold unit" offer many advantages, for instance, possible fault location and isolation [15] including the benefit of short input sequences [12,13,14], the need for a reference or "gold unit" limits its applicability for on-line testing of digital circuits. A new testing method which does not require prior test generation has recently been implemented in some test equipment [8]. In this method, random inputs are fed into the circuit and the averages of the outputs (one count or transition count) are obtained. These averages

are compared with simulated averages or averages obtained from a fault-free circuit. Based on the comparison, the circuit passes or fails the test. The Fluke Trender 1000 Logictester is an example of commercial testers that use this method. This test method is known in the literature as referenceless random testing [8]. Since this method does not require a reference or "gold unit" and only knowledge of input and output statistics is required, the application of this method to on-line testing of digital circuits seems possible.

In evaluating the effectiveness of referenceless random testing, Losq [8] used the following measures of effectiveness.

- Probability of false alarm: the probability that a fault-free circuit fails the test.
- Probability of escape: the probability that a faulty circuit passes the test.

The probability of false alarm depends only on the fault-free output statistics, the experiment length, N , and the test stringency, ϵ . The test stringency defines an acceptance window

$$A = [z_j - \epsilon, z_j + \epsilon],$$

where z_j is the fault-free statistic, for which the circuit passes the test. Losq obtained an upper bound for the probability of false alarm which depended on the test length and stringency. It was found that this probability decreases dramatically as the test length and/or the test

stringency were increased ($\text{prob}(\text{false alarm}) \leq \text{erfc}(\epsilon\sqrt{2N})$). The escape probability, however, is difficult to obtain. Losq presented statistical fault models for large complex combinational circuits and Read Only Memories (ROMs) and approximated the escape probability for these circuits. His results indicated that referenceless random testing is an efficient method for testing combinational circuits and ROMs. The evaluation of the probability of escape for digital sequential circuits is difficult and has not been reported. This research will attempt to tackle this problem.

To obtain the escape probability for any digital circuit, the fault density function must be derived. This function, $\phi(z)$, is the density of faulty circuits having the same output statistic under specified input statistics. The function depends on the internal structure and the statistical fault model of the circuit. Since we are interested in the on-line testing of fielded electronic equipment, i.e., equipment that has passed design and manufacturing acceptance tests, and since accurate evaluation of the escape probability is required, an adequate fault model must be chosen. One such model, known as the pin-package fault model, has been evaluated by Ketelson [6]. According to Ketelson, the pin fault model "appears to retain an acceptable degree of fault coverage while allowing fault analyses and test generation to remain within the realm of computational feasibility." Furthermore, he states that "available failure data for digital integrated

circuits shows the dominant failure mechanism to be 'dead bond' failure." Failures also occur at the interconnections between integrated circuits and between integrated circuit package pins and the silicon chip itself [6]. Electrical overloading also causes failures, for example, overshoots on a pin may damage input or output transistors. All of these failures are implied and included in the pin-package fault model.

A statistical pin-package fault model will be used in this report for the derivation of the fault density function and, therefore, the probability of escape of modular digital circuits.

1.4 Research Objectives and Approach

The purpose of this study is to develop procedures to accurately derive the fault density function of modular digital sequential circuits using the statistical pin-package fault model. This function is used in the *a priori* analyses of the efficiency of referenceless random testing in Built-In-Test (BIT).

The study begins with a development of procedures for relating the output and state probabilities of sequential circuits to the input signal probabilities. Several methods are analyzed. Next, a simple single-input four-state sequential circuit is examined to determine testability measures useful in evaluating the efficiency of referenceless random testing for the detection of faults in the circuits

(see Chapter 3). In Chapter 4, procedures for evaluating the fault density function for various digital sequential circuits under the statistical pin-package fault model assumption are given. Chapter 5 presents results showing the escape probability as a function of input statistics for two types of sequential circuits. These results are used to set up simulation experiments, the results of which are also included in the chapter. Finally, in Chapter 6, conclusions are made and recommendations are suggested for future research.

The appendices describe programs used to compute the escape probabilities, to simulate experiments, and to obtain three-dimensional plots.

2 STEADY-STATE PROBABILITIES OF SEQUENTIAL CIRCUITS

2.1 Introduction

In this chapter methods are presented for obtaining the steady-state state probabilities of sequential circuits. First, Markov chains are analyzed since sequential circuits under random inputs behave as first order Markov chains. Secondly, regular expressions are reviewed and a method for obtaining steady-state probabilities from regular expressions is given. Next, methods for obtaining the steady-state probabilities directly from the circuit are examined. One method takes advantage of iterative networks. The other method is more general and shows how the regular expression of a state or output can be obtained directly from the circuit. From this regular expression, the steady-state probability can be derived. Finally, each method is compared and the situations where each is better applicable are discussed.

2.2 Markov Chain Model of Sequential Circuits

Deterministic sequential circuits under random inputs possess the Markov property that the probability of being in a given state at time n depends only on the state of the circuit at time $n-1$ and the probabilities of the input vector at time $n-1$. Therefore, the behavior of sequential circuits under the application of random inputs can be modeled as a finite first order Markov chain. The chain

is ergodic if the circuit is strongly connected, i.e., it is possible to reach any state from a given state. This requires that for ergodic Markov chains, the steady-state probabilities are non zero. Finally, the Markov chain is stationary if the input probability distributions are independent of time. Hence, in order to analyze the probabilistic behavior of sequential circuits we present a brief development of Markov chain theory. The definitions are taken from [12] with modifications in nomenclature.

Definition: The state probability $\pi_j(n)$ of state S_j at time n is the probability that the circuit is in state S_j after the n^{th} input. The state probability vector $\pi(n)$ is a row vector whose j^{th} element is the state probability.

The sum of the elements of a probability vector must equal one. The state probability vector lists the state probability distribution at time n .

Definition: The n -step transition probability $P_{ij}(n)$ is the conditional probability of entering state S_i . The transition matrix P is the square matrix whose entry at the j^{th} row and i^{th} column is $P_{ij}(1)$ (the one-step transition probability).

The one-step transition probabilities for a sequential circuit are obtained by summing the probabilities of the input vectors that cause a transition from S_i to S_j . Each row in the transition matrix must sum to one.

Each state probability $\pi_j(n)$ is described by the equation

$$\pi_j(n) = \sum_i \pi_i(n-1)P_{ij}(1) \quad (2.1)$$

Equation (2.1) in matrix form is

$$\pi(n) = \pi(n-1)P \quad (2.2)$$

By repeated application of (2.2)

$$\pi(n) = \pi(0) (P \cdots P (P(P))) = \pi(0)P^n \quad (2.3)$$

The vector $\pi(0)$ is the initial state-probability vector. The entry at the j^{th} row and its column of P^n is the n -step transition probability $P_{ij}(n)$. This concludes the development as presented by [13]. We provide below Example 1 [2] in order to help clarify the ideas.

Example 1: In the sequential circuit described by the state diagram of Figure 2.1, the transition matrix is

$$P = \begin{matrix} & \begin{matrix} S_1 & S_2 & S_3 & S_4 \end{matrix} \\ \begin{matrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{matrix} & \begin{bmatrix} q & p & 0 & 0 \\ 0 & 0 & p & q \\ p & q & 0 & 0 \\ 0 & 0 & q & p \end{bmatrix} \end{matrix}$$

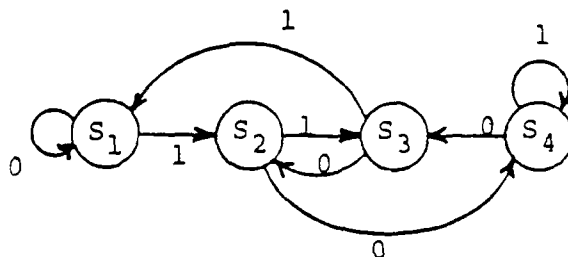


Figure 2.1 State diagram for example sequential circuit

where p is the probability of the input being a logical 1 and $q=1-p$. Assume $\pi(0)=[0.3 \ 0.4 \ 0.2 \ 0.1]$ then a straightforward calculation gives

$$\pi(1)=\pi(0)P=[0.22 \ 0.28 \ 0.34 \ 0.16]$$

$$\pi(2)=\pi(1)P=\pi(0)P^2=[0.316 \ 0.244 \ 0.256 \ 0.184]$$

$$\pi(3)=\pi(0)P^3=[0.268 \ 0.304 \ 0.232 \ 0.196]$$

$$\vdots$$

$$\pi(10)=\pi(0)P^{10}=[0.249 \ 0.251 \ 0.254 \ 0.246]$$

$$\pi(\infty)=\pi(0)P^{\infty}=[0.250 \ 0.250 \ 0.250 \ 0.250]$$

The above calculations illustrate two interesting facts. First we observe that for sufficiently large n , $\pi(n)$ becomes independent of the initial state distribution $\pi(0)$. Second, the state distributions fluctuate about their steady state value. These observations lead to a study of the dynamical properties of a Markov process [2]. Below it is shown that for ergodic Markov chains the steady state distribution of probabilities is independent of the initial state.

2.3 Steady-State Probabilities

Since the transition matrix P is non-singular (the process is ergodic), we obtain for P^n

$$P^n = T \Lambda^n T^{-1} \quad (2.4)$$

where T is the matrix of left eigenvectors of P and Λ is a diagonal matrix with diagonal elements λ_j the eigenvalues of P . In general each eigenvalue of a matrix B with positive or zero elements satisfies at least one of the inequalities

$$|\lambda - b_{ii}| \leq r_i, \quad r_i = \sum_{j=1}^n {}' b_{ij} \quad (2.5)$$

where the prime indicates the term $i=j$ in the sum is omitted. Since the rows of the matrix P sum to one, using (2.5) we conclude that

$$\lambda_j \leq 1 \quad (2.6)$$

where λ_j are the eigenvalues of P . We are interested in the limit

$$\tilde{P} = \lim_{n \rightarrow \infty} P^n = T \lim_{n \rightarrow \infty} \Lambda^n T^{-1}. \quad (2.7)$$

From (2.6) we conclude that $\lim_{n \rightarrow \infty} \Lambda^n = \tilde{\Lambda}$ is independent of n . Thus P possesses a limit as $n \rightarrow \infty$. Furthermore, the limit converges as quickly as the largest Λ_j for $\Lambda_j \neq 1$. Now, we have

$$\tilde{P} = \lim_{n \rightarrow \infty} P^{n+1} = P \lim_{n \rightarrow \infty} P^n = P\tilde{P}. \quad (2.8)$$

For a k state process, (2.8) will be satisfied if \tilde{P} has the form

$$\tilde{P} = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \cdots & \pi_k \\ \pi_1 & \pi_2 & \pi_3 & \cdots & \pi_k \\ \vdots & \vdots & \vdots & & \vdots \\ \pi_1 & \pi_2 & \pi_3 & \cdots & \pi_k \end{bmatrix} \quad (2.9)$$

where π_i is a nonzero constant and $\sum_{i=1}^k \pi_i = 1$. If $\pi(0)$ is the initial state distribution then the steady state distribution π can be written

$$\pi = \lim_{n \rightarrow \infty} \pi(0)P^n = \pi(0)\tilde{P} = [\pi_1, \pi_2, \cdots, \pi_k].$$

Hence, the rows of P correspond to the steady-state value of the state probability vector. To obtain π without taking the limit $\lim_{n \rightarrow \infty} P^n$ we note that

$$\pi = \lim_{n \rightarrow \infty} \pi(n) = \lim_{n \rightarrow \infty} \pi(n-1)P = \pi P. \quad (2.10)$$

Using (2.10), the following system of equations is obtained:

$$\begin{aligned} \pi_1 P_{11} + \pi_2 P_{21} + \cdots + \pi_k P_{k1} &= \pi_1 \\ \pi_1 P_{12} + \pi_2 P_{22} + \cdots + \pi_k P_{k2} &= \pi_2 \\ \vdots & \\ \pi_1 P_{1k} + \pi_2 P_{2k} + \cdots + \pi_k P_{kk} &= \pi_k \end{aligned} \quad (2.11)$$

where the P_{ij} are the one-step transition probabilities. Since one equation of (2.11) can be found from a linear combination of the other equations, the system of equations must include the equation

$$\sum_{i=1}^k \pi_i = 1. \quad (2.12)$$

The example which follows illustrates this procedure and derives steady-state probabilities from a state diagram.

Example 2

The state diagram of a J-K flip-flop is shown in Figure 2.2(b).

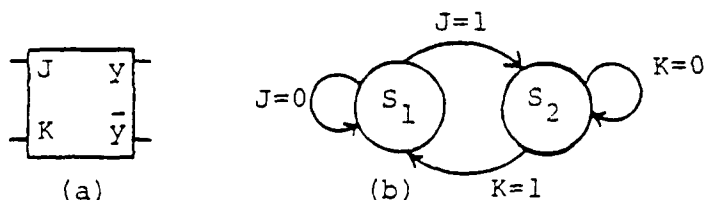


Figure 2.2 J-K flip-flop

The eigenvalues of P are $\lambda_1=1$ and $\lambda_2=1-(j+k)$. The eigenvectors corresponding to λ_1 and λ_2 are $t_1=\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $t_2=\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ respectively. Using (2.4) we have

$$P^n = \begin{bmatrix} 1 & 1 \\ 1 & 1 - \frac{k}{j} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1-(j+k) \end{bmatrix}^n \begin{bmatrix} \frac{k}{j} & 1 \\ 1 & -1 \end{bmatrix} \frac{j}{j+k}.$$

Taking the limit of P^n as $n \rightarrow \infty$ we have,

$$\begin{aligned} \tilde{P} &= \lim_{n \rightarrow \infty} P^n = \begin{bmatrix} 1 & 1 \\ 1 & 1 - \frac{k}{j} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{k}{j} & 1 \\ 1 & -1 \end{bmatrix} \frac{j}{j+k} \\ \tilde{P} &= \begin{bmatrix} \frac{k}{j+k} & \frac{j}{j+k} \\ \frac{k}{j+k} & \frac{j}{j+k} \end{bmatrix} \end{aligned}$$

Thus $\pi_1 = \frac{k}{j+k}$ and $\pi_2 = \frac{j}{j+k}$ are the steady state probabilities of states s_1 and s_2 , respectively. As an alternative, using (2.11) and (2.12) we can write

$$(1-j)\pi_1 + k\pi_2 = \pi_1$$

$$\pi_1 + \pi_2 = 1$$

from which $\pi_1 = \frac{k}{j+k}$ and $\pi_2 = \frac{j}{j+k}$ as before. We can now apply these ideas in Example 3 to a more realistic and complex sequential circuit in order to demonstrate the power of the technique.

Example 3

The state diagram of a 3-bit synchronous counter similar to the 74163 four-bit counter excluding the

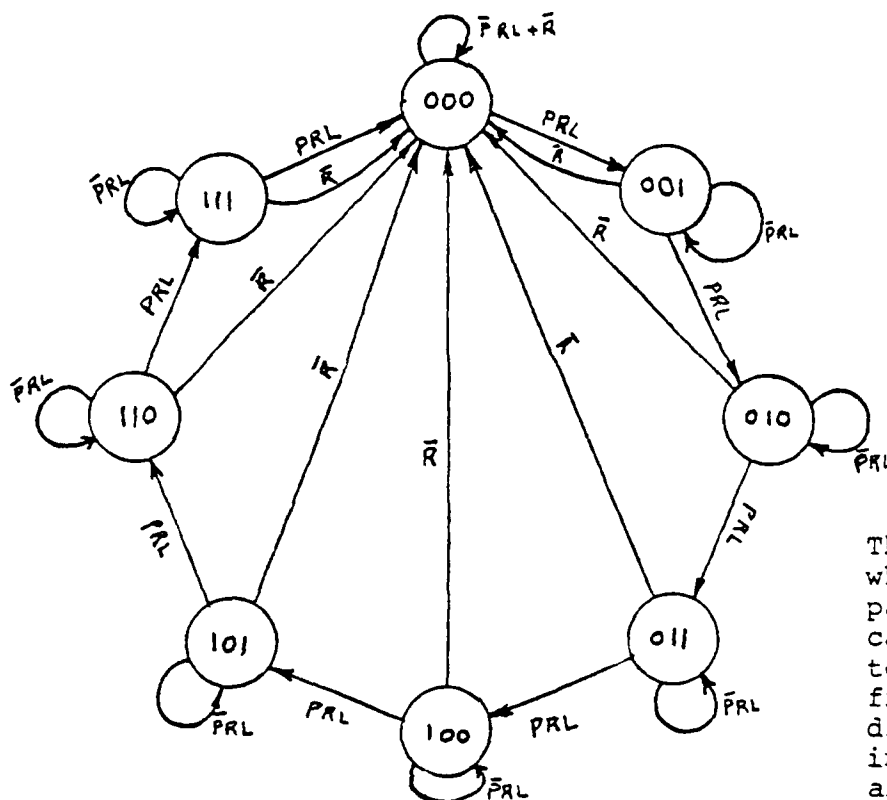
programmable inputs is shown in Figure 2.3. A block diagram of the device and a table of input definitions is shown in Figure 2.4.

Let s_i , $i=0,7$ denote the steady-state probabilities of the states and r , l , p denote the probabilities of R , L , and P_t . From the state diagram according to (2.11) and (2.12) and taking into account the preset inputs A , B , and C with probabilities a , b and c we can write the system of equations

$$\begin{aligned} s_0 &= s_0 \bar{p} \bar{l} r + \bar{l} r \bar{a} \bar{b} \bar{c} + \bar{r} + p \bar{l} r s_7 \\ s_1 &= s_1 \bar{p} \bar{l} r + \bar{l} r a \bar{b} \bar{c} + p \bar{l} r s_0 \\ s_2 &= s_2 \bar{p} \bar{l} r + \bar{l} r \bar{a} b \bar{c} + p \bar{l} r s_1 \\ &\vdots \\ s_6 &= s_6 \bar{p} \bar{l} r + \bar{l} r \bar{a} b c + p \bar{l} r s_5 \\ s_7 &= s_7 \bar{p} \bar{l} r + \bar{l} r a b c + p \bar{l} r s_6 . \end{aligned}$$

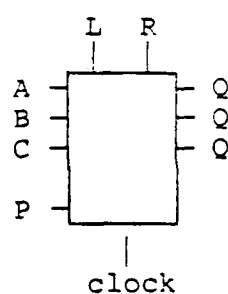
The above equations lead directly to a generalization for M -bit programmable synchronous circuits. We can write then, making the substitutions $\bar{p} \bar{l} r = \alpha$, $p \bar{l} r = \beta$, $\bar{l} r = \gamma$, and i_k for the input combinations A , B , and C (e.g., $i_4 = \bar{a} \bar{b} c$), $(N=M-1)$,

$$\begin{aligned} s_0 &= \alpha s_0 + \beta s_7 + \gamma i_0 + \bar{r} \\ s_1 &= \alpha s_1 + \beta s_0 + \gamma i_1 \\ s_2 &= \alpha s_2 + \beta s_1 + \gamma i_2 \\ &\vdots \\ s_N &= \alpha s_N + \beta s_{N-1} + \gamma i_N \\ \sum_{i=0}^N s_i &= 1 . \end{aligned}$$



The many paths which represent parallel load cases are omitted for clarity from the state diagram but included in the analysis.

Figure 2.3 State diagram 3-bit synchronous counter



R = low, reset
 p = high, count enable
 P = low, count disable
 L = low, load from inputs A,B,C

Figure 2.4 Synchronous counter

Solving for s_n we have

$$s_n = \frac{\beta^n s_0}{(1-\alpha)^n} + \gamma \sum_{k=1}^n \frac{\beta^{n-k} i_k}{(1-\alpha)^{n+1-k}} \quad n \neq 0, n=1, \dots, N$$

$$s_0 = 1 - \sum_{n=1}^N s_n.$$

Substitution and solving for s_0 we obtain

$$s_0 = \frac{1-\gamma \sum_{n=1}^N \sum_{k=1}^n \beta^{n-k} i_k / (1-\alpha)^{n+1-k}}{1 + \sum_{n=1}^N \frac{\beta^n}{(1-\alpha)^n}}$$

$$s_j = \frac{\beta^j}{(1-\alpha)^j} \frac{1-\gamma \sum_{n=1}^N \sum_{k=0}^n \beta^{n-k} i_k / (1-\alpha)^{n+1-k}}{1 + \sum_{n=1}^N \frac{\beta^n}{(1-\alpha)^n}}$$

$$+ \gamma \sum_{k=1}^j \frac{\beta^{j-k} i_k}{(1-\alpha)^{j+1-k}} \quad j \neq 0.$$

Making the substitution $\lambda = \frac{\beta}{1-\alpha}$ we finally obtain

$$s_0 = \frac{1 - \frac{\gamma}{1-\alpha} \left(\frac{1}{1-\lambda} \right) \sum_{k=1}^{N-1} i_k (1-\lambda)^{N-k+1}}{1 + \sum_{n=1}^N \lambda^n} \quad (2.13)$$

$$s_j = \lambda^j s_0 + \frac{\gamma}{1-\alpha} \sum_{k=1}^j \lambda^{j-k} i_k \quad j=1, 2, \dots, N. \quad (2.14)$$

Equations (2.13) and (2.14) are general equations describing the steady-state probabilities for an n -bit counter with a configuration corresponding to Figure 2.4.

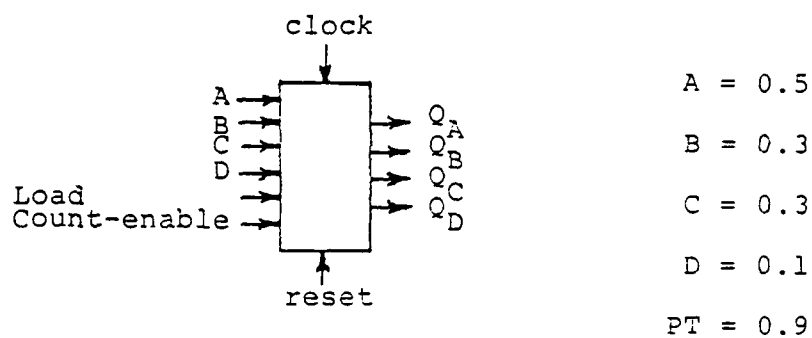
Using (2.13) and (2.14) it is now possible to describe the probability behavior of the states of the counter as a function of its input probabilities.

In Figure 2.5 we present a three-dimensional plot of the probability of state S_1 versus the LOAD and RESET input probabilities for a four-bit synchronous programmable counter. State S_1 corresponds to $Q_A=1$, $Q_B=0$, $Q_C=0$, and $Q_D=0$. The input probabilities for A,B,C,D, and PT are held constant as indicated in the figure.

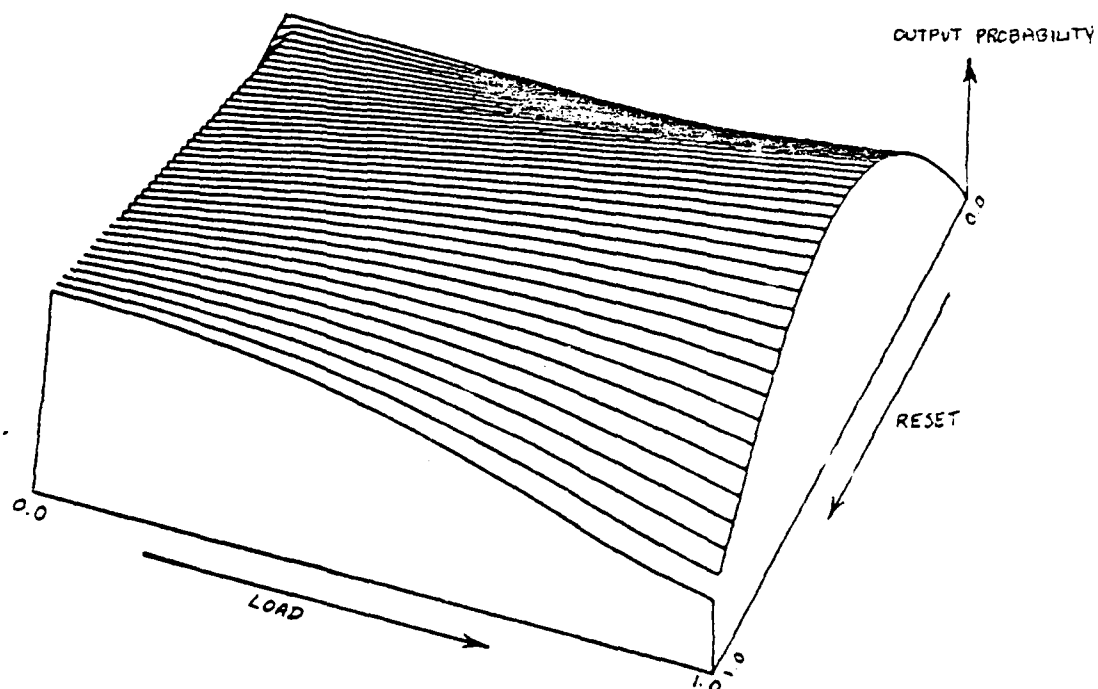
We introduce at this point the usefulness of these ideas with respect to fault detection. Figure 2.6 shows the three-dimensional plot of state S_1 with preset input C stuck-at-one ($\text{prob}(c)=1.0$). Comparing Figures 2.5 and 2.6, we can observe the change in state probability due to a single stuck-at-fault.

In this section we have presented a concise development of Markov chains necessary for analyzing the probabilistic behavior of sequential circuits. Furthermore, we have illustrated with an original example the power of Markov chain analysis by deriving the steady-state state probabilities of a general n-bit synchronous counter.

Considering the extensive literature available on Markov chains, it would seem that the introduction of new methods for the computation of steady-state probabilities is useless. In this research, however, we are interested in efficient methods. Although the system of equations for the steady-state probabilities can be written down directly

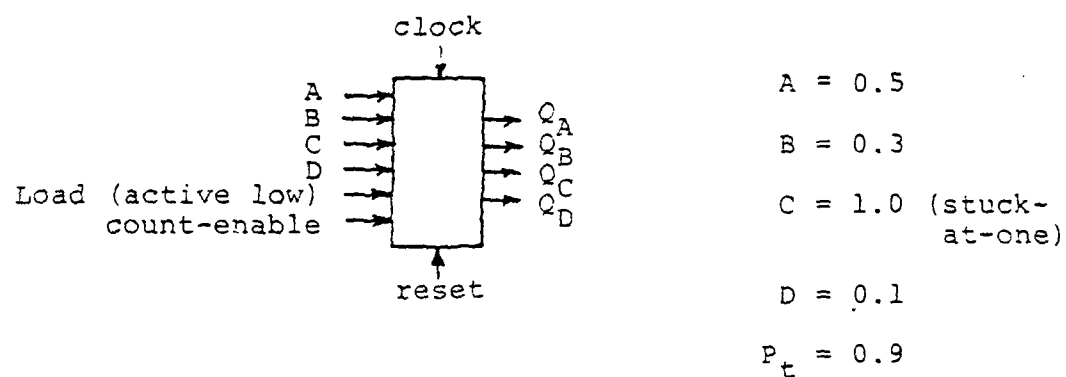


a. counter package and constant input probabilities



b. probability of S_1 vs. LOAD and RESET

Figure 2.5 Output probability versus input probability for a synchronous counter



a. counter package and constant input probabilities

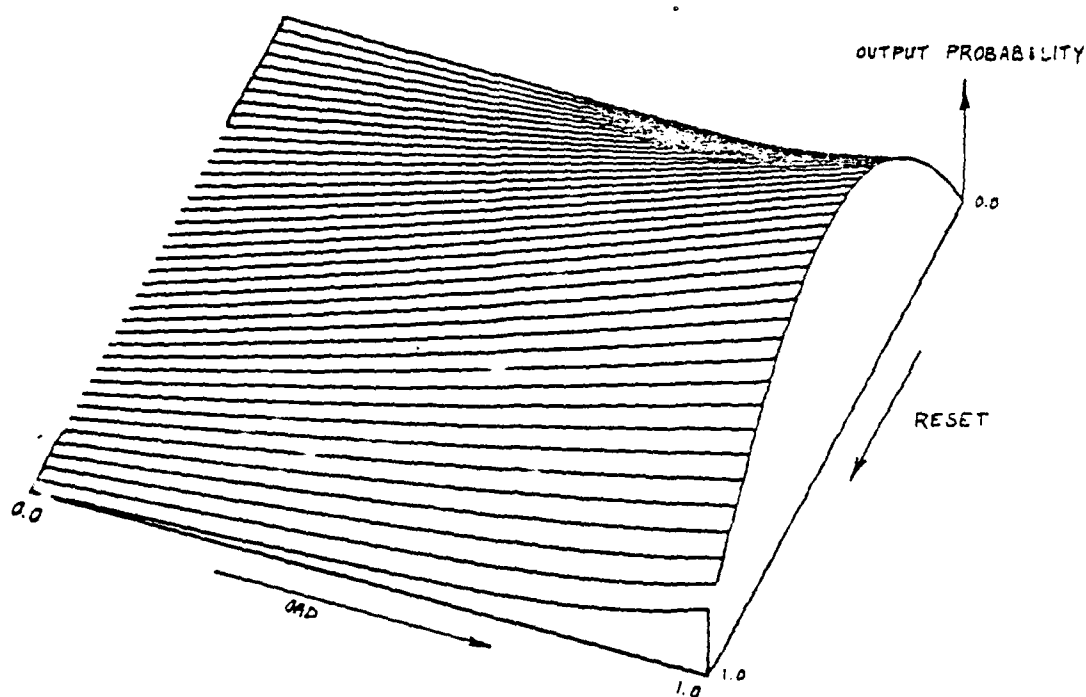
b. probability of S_1 vs. LOAD and RESET

Figure 2.6 Output probability with a failure, c-stuck-at-one

from the state diagram, according to Markov chain analysis, the solution of these equations especially where formulae are required is difficult and not straightforward. In the next section a convenient method is presented where the state probabilities are directly found from the regular expression of a state expressed in synchronized form. Furthermore, using the Markov chain analyses it is not possible to obtain the probabilities directly from the circuit. A method, however, exists for deriving the regular expression of states and outputs directly from the circuit. From these expressions it is possible to directly write down the input/output probability equation. The next section, therefore, introduces regular expressions and shows how the steady-state probabilities can be obtained from these expressions.

2.4 Steady-State Output Probabilities from Regular Expressions

2.4.1 Introduction

A method which can potentially provide substantial savings in the computation of output probabilities is the regular expression investigated by Parker and McCluskey [11]. This approach requires the existence and knowledge of synchronizing sequences for the machines. This limits the usefulness of the approach since these sequences do not exist for all machines. In addition, the presence of a

fault may alter a machine's synchronizing sequence resulting in very tedious fault analysis.

However, when regular expressions are derived directly from the sequential circuits [11] they can be put in a form such that the output probability can be easily computed. In the next section regular expressions are reviewed and examples are provided. The concept of the derivative of a regular expression is also examined. Finally, the method for obtaining the steady-state probabilities for regular expressions is analyzed. The ideas developed in these sections will be used in discussing the derivation of output probabilities directly from sequential circuit implementations.

2.4.2 Review of regular expressions

A background in regular expressions is given below which is taken from the paper by Parker and McCluskey [11]. Further discussions of regular expressions can be found in textbooks such as Kohavi [7] and Booth [2]. Booth provides a useful list of references for further study of the topic.

Regular expressions are defined recursively as follows: given an alphabet $A_k[a_0, a_1, a_2, \dots, a_{k-1}]$, the symbols λ and ϕ , and the regular operators '+', '.', and '*' and parentheses,

- 1) a string consisting of a single alphabet symbol, single λ , or single ϕ is a regular expression.
- 2) If P and Q are regular expressions, then so are $(P+Q)$, $(P \cdot Q)$, and P^* .

- 3) No other string of symbols is a regular expression unless it follows from (1) and (2) in a finite number of steps.

Regular expressions represent sets of sequences of symbols. The symbol λ represents the sequence of zero length. The symbol ϕ represents the null or empty set of sequences. The operation '+' in $P+Q$ denotes the union of the sets P and Q . The operation '.' in $P \cdot Q$ (from now on written as PQ with "." understood) is the concatenation elements P and Q . In general, concatenation is not commutative, i.e., PQ may not equal QP . The star operator '*' or iterate of a set is defined as the infinite union

$$p^* = \lambda + P + PP + PPP + \dots, \quad (2.15)$$

which we abbreviate as

$$p^* = \bigcup_{i=0}^{\infty} p^i \quad (2.16)$$

where p^i is i copies of P concatenated and p^0 is defined as λ . Given that P , Q , and R are regular expressions, then the following properties hold

$$P+Q = Q+P \quad (+ \text{ commutative}) \quad (2.17)$$

$$(P+Q)+R=P+(Q+R) \quad (+ \text{ associative}) \quad (2.18)$$

$$(PQ)R=P(QR) \quad (\cdot \text{ associative}) \quad (2.19)$$

$$PQ + PR = P(Q+R) \quad (\text{left distributivity}) \quad (2.20)$$

$$PR + QR = (P+Q)R \quad (\text{right distributivity}) \quad (2.21)$$

$$R+\phi = \phi + R = R \quad (+ \text{ identity}) \quad (2.22)$$

$$R\phi = \phi R = \phi \quad (\cdot \text{ zero}) \quad (2.23)$$

$$R\lambda = \lambda R = \underline{R} \quad (\cdot \text{ identity}) \quad (2.24)$$

$$R + R = R \quad (\text{idempotency}) \quad (2.25)$$

$$\lambda^* = \lambda \quad (2.26)$$

$$\phi^* = \lambda \quad (2.27)$$

Definition 2.1: The derivative [3] of a regular expression R with respect to a sequence a , denoted $D_a R$, is another regular expression given by

$$D_a R = \{s \mid as \in R\} \quad (2.28)$$

Also,

$$D_{ab} R = D_b (D_a R). \quad (2.29)$$

Definition 2.2: The symbol I is reserved for the union of the alphabet symbols and constitutes a DON'T CARE symbol of length 1. Then I^* is the set of all possible strings. This completes the background on regular expressions obtained from Parker and McCluskey [11].

Consider, as an example, the machine described by the state diagram of Figure 2.7. The input alphabet is $A_2 = \{0,1\}$. The regular expression description of the machine is

$$R = (0+1)^* 10 (10+00)^*. \quad (2.30)$$

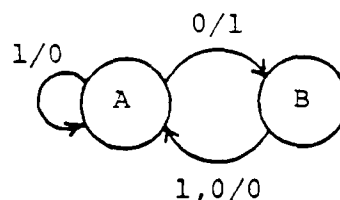


Figure 2.7 State diagram

The regular expression (2.30) states that the machine will have an output whenever the sequence 10 is applied to the input regardless of the previous inputs. The sequence 10 is thus a synchronizing sequence for the machine.

Also, when the sequence 10 is applied to the machine, the machine will continue to produce an output for any combination of the union $10+00$ following the sequence. These are easily derived and verified from the state diagram of the machine. We will be interested in writing down the steady state output probabilities of the machine of Figure 2.7 directly from the regular expression (2.30). We will need, however, to introduce the following definitions again taken from Parker and McCluskey [11].

Definition 2.3: A sequence s is defined to be a substring of a regular expression R , denoted as $s \in \text{sub}(R)$, iff there exists an $n \in I^*$ such that $D_{ns}R \neq \emptyset$. If $s \in \text{sub}(R)$ then for some $r \in R$, the sequence is embedded in r . For example, 011 is a substring of 110110, since there exists an $n \in I^*$ ($n=11$) such that $D_{11011}(110110) \neq \emptyset$. However, 111 is not a substring of 110110.

Definition 2.4: A simple expression (or elementary expression) is the concatenation of zero or more alphabet symbols.

Definition 2.5: A finite composite expression is the union of a finite number of simple expressions. Example: $T=ab+bc+acb$ is a finite composite expression.

Definition 2.6: An infinite composite expression is the star iterate of a finite composite expression. Example: $T=(a+ab+abca)^*$ is an infinite composite.

Definition 2.7: A concatenate composite expression is the concatenation of two or more finite or infinite composite expressions. For example, let A, B be finite composite, C be infinite composite. Then ABC , $A*B$, AB^* , and $A*B^*$ are concatenate composities, as is AB , but AB by distributivity reduces to another finite composite.

Definition 2.8: All other expressions that are not simple, finite/infinite/concatenate composite in form are called complex expressions. For example, $(ab*c+a*b)^*$ is complex.

Definition 2.9: A delay expression is a special case of concatenate composite expression of the form $T=I*S$ for some finite composite S .

Definition 2.10: A synchronized expression is of the form $I*SR$ where S is finite composite and R is a regular expression. Delay expressions are a special case. The name "synchronized" stems from the machine realization, where S represents a set of synchronizing sequences [11]. To determine the acceptability of a synchronized expression, all symbols back to the last occurrence of $s \in S$ must be examined.

Definition 2.11: Given an alphabet $I=a_0, a_1, \dots, a_{k-1}$, define the probability $P(a_i)$ of the symbols as a set of real numbers such that $0 \leq P(a_i) \leq 1$. Furthermore, define $P(\lambda)=1$ and $P(\phi)=0$.

Lemma 2.1: The probability of a simple expression of length $m \geq 0$ is given by the product of the probabilities of its symbols. For example, for $E=abc$, the probability of this expression is $P(a)P(b)P(c)$. We are assuming that all appearances of any symbol at any place in an event are independent of any other symbol.

2.4.3 Steady-state probabilities

We examine the synchronized expression of definition 2.10 given in (2.31). The English description of a recognizer of (2.31) is [11] the machine that accepts any string ending

$$R = I*ST^* \quad (2.31)$$

in S followed by zero or more occurrences of T . Parker and McCluskey introduce an improper machine, shown in Figure 2.8, that recognizes $I*ST^*$. In the machine, V is the set of all strings that do not contain an embedded S . Also, U is the set of all strings that do not contain an embedded S or T . They then use concepts in steady-state Markov chain analysis to arrive at the steady-state output probability of the machine of Figure 2.8. Thus, if $s=P(S)$ and $t=P(T)$ we can write

$$P_B = (s+t)P_B + sP_A \quad (2.32)$$

$$P_B + P_A = 1.0. \quad (2.33)$$

Solving the above equations for P_B we obtain

$$P(\text{state } B) = P(I*ST^*) = s/(1-t). \quad (2.34)$$

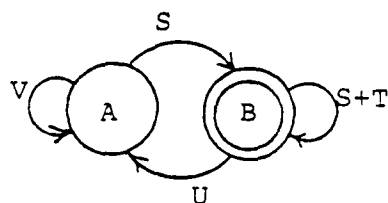


Figure 2.8 Improper machine

Parker and McCluskey thus give and prove the following theorem.

Theorem 2.1: Given simple S and T with $T \neq \lambda$, $s \notin \text{sub}(T^*)$, $E = I^*ST^*$, $s = P(S)$, and $t = P(T)$, then $P_\infty(E) = s/(1-t)$.

Example: Referring back to the machine of Figure 2.7 we note that the regular expression (2.30) describing the machine is a synchronized expression. We thus apply Theorem 2.1 and obtain $(P(S) = p(1-p), P(T) = (1-p)^2)$,

$$P(\text{state } B) = P[I^*(01)(00)^*] = \frac{P(1-p)}{1-(1-p)^2} = \frac{1-p}{2-p}. \quad (2.35)$$

The above result can also be obtained by steady-state Markov analysis:

$$P_B = (1-p) P_A \quad (2.36)$$

$$P_B + P_A = 1 \quad (2.37)$$

$$P_B = (1-p)/(2-p). \quad (2.38)$$

We have shown how the steady-state probabilities of a sequential circuit can be obtained from regular expression descriptions of the states and outputs expressed in synchronized form.

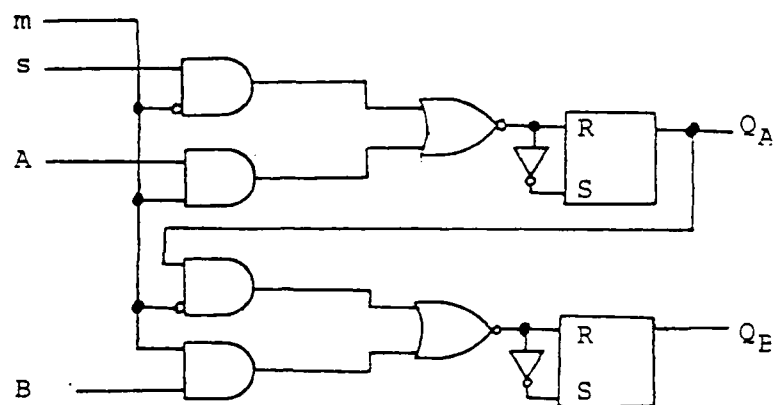
Through an example we illustrate the straightforwardness of the procedure and the advantage of the method over

the normal Markov method of solving a system of equations. The disadvantages of the method are associated with the difficulty in obtaining regular expression descriptions of sequential circuits in synchronized form. Synchronizing sequences and sequences that return the machine to a given state are not easily derived from a state diagram description of the machine. As in the Markov analysis, using this method output probabilities are obtained independent of the actual implementation of the sequential circuit. However, a method exists which is discussed below for which the regular expression is derived directly from the circuit. The advantage of using the circuit is that resulting regular expression is in synchronized form. Hence, the applicability of the derivation of steady-state probabilities from regular expressions.

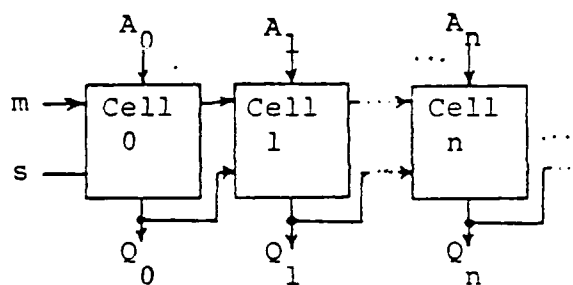
2.5 Derivation of Output Probabilities of Iterative Sequential Circuits

Due to their structure, iterative sequential circuits simplify the derivation of the output probabilities directly from the circuit. In this section, we will investigate this procedure for obtaining output probabilities by way of an example.

Figure 2.9a shows the circuit diagram of a parallel load, right-shift, 2-bit shift register. In Figure 2.9b the iterative structure of the circuit is illustrated. The circuit diagram for each cell is shown in Figure 2.10.



(a)



(b)

Figure 2.9 Logic diagram of shift register

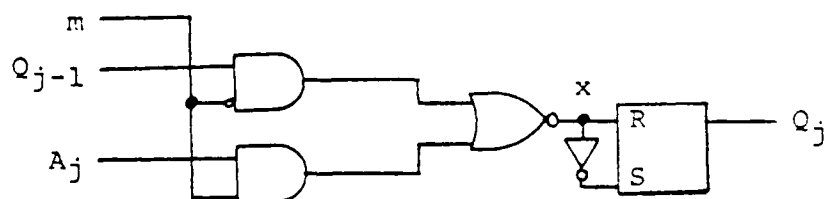


Figure 2.10 Cell logic diagram

The output probability of the cell as a function of the input probabilities can be found as follows.

Consider the point X in Figure 2.10. The probability of X is obtained through the combinational network and is given by

$$P_x = 1 - q_{j-1} + q_{j-1}m + m a_j$$

where q_{j-1} is the shift input probability from the previous cell, m is the mode control probability and a_j is the parallel load probability for that cell.

The probability of the cell output, Q_j , can be found using the steady-state Markov analysis, from the state diagram of the flip-flop shown in Figure 2.11.

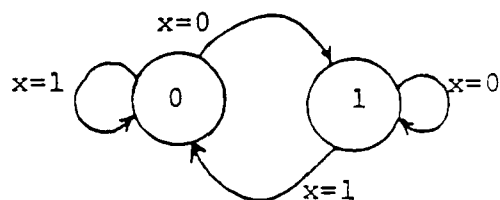


Figure 2.11 Flip-flop state diagram

Thus,

$$q_j = (1-x)q_j - (1-x)(1-q_j) = 1-x$$

where q_j is the output probability. Substituting for x we obtain

$$q_j = 1 - (1 - q_{j-1} + q_{j-1}m + m a_j) = q_{j-1}(1-m) + m a_j.$$

Thus, the output probability of a cell can be expressed as a function of the inputs for the cell and the output of the

previous cell. The general expression of a cell output as a function of the shift input s , model control m , and preset probabilities a_j ($j=0, \dots, n-1$) can be derived as shown below.

$$q_0 = s\bar{m} + ma_0$$

$$q_1 = (s\bar{m} + ma_0)\bar{m} + ma_1 = s\bar{m}^2 + m\bar{m}a_0 + ma_1$$

$$q_2 = (s\bar{m}^2 + m\bar{m}a_0 + ma_1)\bar{m} + ma_2 = s\bar{m}^3 + m\bar{m}^2a_0 + m\bar{m}a_1 + ma_2$$

and in general

$$q_j = s\bar{m}^{j+1} + m \sum_{k=0}^j \bar{m}^k a_{j-k}, \quad (\bar{m}=1-m). \quad (2.39)$$

Note that in the iterative structure above, the cell outputs Q_j depends only on Q_{j-1} and not on Q_{j-1} . This cell structure, shown in Figure 2.9b, allows for the simple calculation of cell output probabilities as illustrated by the example above. This is because in the cell Boolean output function

$$Q_j = f_j(x_0, x_1, \dots, x_n, Q_{j-1})$$

the random inputs x_i ($i=0, \dots, n$) and the random output Q_{j-1} are independent. This is true if the current random value on an input line is independent of the previous random values on the line. Because of this property, the method of obtaining output probabilities from input probabilities for combinational circuits can be used [9]. This justifies (2.39) in the example above.

The cell outputs, however, are not independent. For instance, in Figure 2.12, Q_j and Q_{j-1} are not independent.

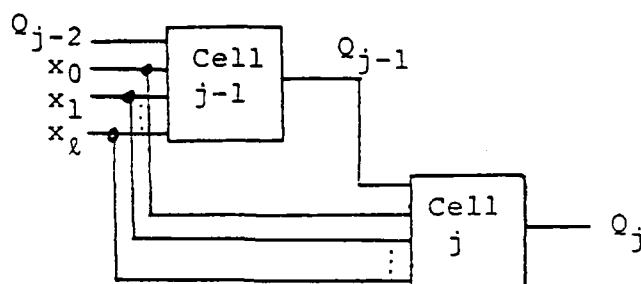


Figure 2.12 Non-feedback cell structure

1) Q_{j-1} and x_i ($i=0,1,\dots,l$) are independent. Since Q_{j-1} is a Boolean function of the previous values of the inputs x_i^{t-1} and since the current values of the inputs x_i^t are independent of previous values, x_i^{t-1} , then Q_{j-1}^t and the inputs x_i^t are independent.

2) Q_j and Q_{j-1} are not independent. Q_j^t is a function of Q_{j-1}^{t-1} and the inputs x_i^{t-1} . Q_{j-1}^t also depends on x_i^{t-1} . Thus since Q_j^t and Q_{j-1}^t both depend on x_i^{t-1} , they are not independent.

Consider the shift register of Figure 2.9. With the state assignment of Table 2.1 and using steady-state Markov analysis, the steady-state probabilities are obtained.

They are:

$$P_A = \bar{s}\bar{m} - s\bar{s}\bar{m}^2 - \bar{s}am\bar{m} + \bar{b}a\bar{m}$$

$$P_B = s\bar{m} - s^2\bar{m}^2 - sam\bar{m} + bam$$

$$P_C = s^2\bar{m}^2 + asm\bar{m} + mab$$

$$P_D = \bar{m}^2s\bar{s} + \bar{m}m\bar{s}a + \bar{b}a\bar{m}$$

where in general $\bar{y}=1-y$ and s , \bar{m} , a , and b are the shift, mode control, and preset input probabilities. Now, the

Table 2.1 State assignment

State	Q_B	Q_A
A	0	0
B	0	1
C	1	1
D	1	0

probabilities of output lines Q_A and Q_B can be written

$$q_A = P_B + P_C = s\bar{m} + ma$$

$$q_B = P_C + P_D = \bar{m}^2s + am\bar{m} + mb.$$

These results can be obtained from (2.3a) by noting that $q_0=q_A$, $q_1=q_B$, $a_0=a$, and $a_1=b$. Therefore, the results obtained by the two separate methods agree.

Taking advantage of the iterative cell structure of a sequential circuit, we derived a general expression for the cell output steady-state probabilities. Given the cell number, the steady-state probability as a function of input probabilities could be obtained directly from the expression. Furthermore, the derivation was directly from the circuit and did not require knowledge of the state diagram of the sequential circuit. Also, the method was straightforward and led to a general result, while the results obtained from the state diagram depended on specifying a fixed number of cells (two in the example), obtaining the state diagram, and finally obtaining the steady-state probabilities from the state diagram. Thus, the method of obtaining output probabilities directly from iterative sequential circuits

is quite powerful and is advantageous over the previous lengthy method of obtaining the steady-state probabilities from the state diagram.

2.6 State Probabilities Directly from Sequential Circuit Using Regular Expressions

In this section regular expressions are derived directly from sequential circuits [4]. From the regular expressions of the states, the state probabilities are derived based on the method presented in Section 2.4.

A sequential circuit is constructed from unit delay elements and logic gates [4]. If flip-flops are used they can be modelled as delay elements with additional combinational logic. The sequential circuit has state variables y_1, y_2, \dots, y_m , where each y_j is the output of the j^{th} unit delay element. Following Bryzowski [4] we first consider circuits with single input, X , and output, Z . The next state and output functions are

$$y_j' = f_j(y_1, y_2, \dots, y_m, X) \quad (2.40)$$

$$Z = g(y_1, y_2, \dots, y_m) \quad (2.41)$$

where f_j and g are Boolean functions.

We shall be concerned with sequences that produce an output $z=1$ at time t . Thus we give the following definition from [4].

Definition: An input sequence s of length $t > 0$ energizes a point A in a sequential circuit C started in state q , if and only if the signal at A is 1 at time t .

The sequence of zero length, λ , energizes all points with a logical 1 signal at time t .

From this definition the following properties are obtained for sequential circuits [4]:

- 1) The set of sequences energizing the input lead is the set of all sequences ending in 1 which is denoted by the regular expression $I1$. (By convention, assume that λ does not energize the input level.)
- 2) Let the set of sequences energizing the j^{th} input $j=1,2,\dots,n$ of a logical gate performing the Boolean function f , be denoted by S_j . Then the set of sequences energizing the output of the gate is given by $f(S_1, S_2, \dots, S_n)$. (The gate is assumed to have no delay.)
- 3) If R is the set of sequences energizing the input of a unit delay, then its output is energized by $R1+\delta(y(1))$, where $\delta(0)=\phi$ (the empty set of sequences) and $\delta(1)=\lambda$.

Figure 2.13 illustrates the above properties.

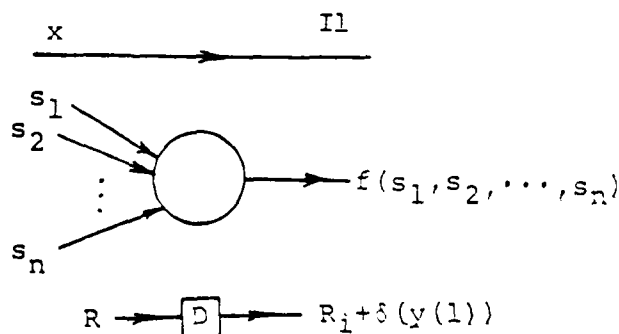


Figure 2.13 Effects of circuit elements [4]

Following Brzozowski, consider a general sequential circuit C characterized by (2.40) and (2.41). Let the set of sequences energizing the input of the j^{th} delay of C be called the j^{th} delay (regular) set, and be denoted by R_j . Let the set of sequences energizing the output level be called the output (regular) set and be denoted by Q_j . Then we can state:

Theorem: The delay and output sets of a sequential circuit C characterized by (2.40) and (2.41) must obey the equations:

$$R_j = f_j[R_1 i + \delta(y_1(1)), R_2 i + \delta(y_2(1)), \dots, R_m i + \delta(y_m(1)), I1] \quad (2.42)$$

$$Q = g[R_1 i + \delta(y_1(1)), R_2 i + \delta(y_2(1)), \dots, R_m i + \delta(y_m(1))] \quad (2.43)$$

If in the initial state all delay outputs are 0, then a simpler form of the above equations results:

$$R_j = f_j(R_1 i, R_2 i, \dots, R_m i, I1) \quad (2.42a)$$

$$Q = g[R_1 i, R_2 i, \dots, R_m i]. \quad (2.43a)$$

Note the one-to-one correspondence of the above equations and (2.40) and (2.41), when R_j is identified with y_j^1 , $R_j i$ with y_j , $I1$ with x , and Q with z [4]. These equations (2.42a) and (2.43a) are called *regular circuit equations* and our problem is to solve these equations. However, we are interested in sequences that are accepted by the circuit.

Definition: A sequential circuit C is said to accept an input sequence S of length $t > 0$ from initial state $q(1)$

if and only if, when C is started in $q(1)$ and S is applied, the output at time $t+1$ is $z=1$. If $z=1$ for the initial state $q(1)$ at time 1, C is said to accept the set λ consisting of the sequence of zero length [4]. According to the above definition we obtain the following corollary [4]:

Corollary: If the delay and output sets of a sequential circuit are given by (2.42a) and (2.43a), then the set P of sequences accepted by the circuit is

$$P = q(R_1, R_2, \dots, R_m). \quad (2.44)$$

In order to solve (2.42a) and (2.44) we need to introduce reverse regular expressions. This is done so that the derivatives of the equations can be taken. Basically, given a circuit C with $z=1$ at time $t+1$, the sequences that "could have been applied to C started in the initial state at some time earlier in order to produce this condition" are given by the reverse regular expression.

Reverse approach to the solution of regular circuit equations: To solve the regular circuit equations we first reverse them. Noting that $r_j = R_j^-$ is used to denote the reverse expression for R_j , we obtain

$$r_j = f_j(ir_1, ir_2, \dots, ir_m, 1I), \quad j=1, 2, \dots, m \quad (2.45)$$

$$P = q(r_1, r_2, \dots, r_j). \quad (2.46)$$

Taking the derivatives with respect to 0 and 1 we obtain

$$D_0 r_j = f_j(r_1, r_2, \dots, r_m) \quad (2.47)$$

$$D_1 r_j = f_j(r_1, r_2, \dots, r_m, I). \quad (2.48)$$

Since the derivatives are Boolean functions of r_1, r_2, \dots, r_j then all derivatives with respect to longer sequences are Boolean functions of the r_j [4]. Moreover, r_j has a finite number of distinctive derivatives and "since there is a finite number of Boolean functions of the r_j , the process of constructing derivatives will terminate without difficulty [4]."

Now, we note that

$$D_0 P = a(D_0 r_1, D_0 r_2, \dots, D_0 r_m), \quad (2.49)$$

$$D_1 P = a(D_1 r_1, D_1 r_2, \dots, D_1 r_m). \quad (2.500)$$

Therefore, there are a finite number of derivatives of P since there are a finite number of derivatives of r_j . Once the process of taking derivatives of P is terminated, using the following properties the reverse regular expression P can be obtained [4].

$$D_s P = \bigcup_k a_k D_{sa_k} P, \quad (2.51)$$

$$P = \bigcup_k a_k D_{a_k} P + \delta(P). \quad (2.52)$$

For a single input circuit $a_k = (0, 1)$ and noting that s represents a sequence of 1s and 0s, we have

$$D_s P = 0 D_{s_0} P + 1 D_{s_1} P, \quad (2.53)$$

$$P = 0 D_0 P + 1 D_1 P + \delta(P). \quad (2.54)$$

In addition to the above two expressions, the following solution to the equation $X = AX + B$, where A, B , and X are regular expressions, is required [4]:

$$X = A * B . \quad (2.55)$$

The procedure for deriving the regular expression of the states of sequential circuits, introduced above, will be illustrated by an example. In the example, the state probabilities will also be derived.

Example 1: Consider the sequential circuit of Figure 2.14. The J-K flip-flop Boolean excitation functions are

$$J_1 = \bar{x} y_2 \quad (2.56a)$$

$$K_1 = 1 \quad (2.56b)$$

$$J_2 = \bar{x} \bar{y}_1 \quad (2.56c)$$

$$K_2 = y_1 y_2 + x y_2 . \quad (2.56d)$$

A J-K flip-flop implemented with a unit delay element is shown in Figure 2.15. The Boolean excitation function for the input to the delay element is

$$y' = \bar{y} J + \bar{K} y . \quad (2.57)$$

If the J-K flip-flops used to implement the sequential circuit were replaced with the equivalent circuit of Figure 2.15, the excitation functions of the delay element inputs will be, by substitution of (2.56) into (2.57) for each delay element,

$$y'_1 = J_1 \bar{y}_1 + \bar{K}_1 y_1 = \bar{x} y_2 \bar{y}_1 \quad (2.58)$$

$$y'_2 = J_2 \bar{y}_2 + \bar{K}_2 y_2 = \bar{x} \bar{y}_1 \bar{y}_2 + \overline{y_1 y_2 + x y_2} y_2 = \bar{x} \bar{y}_1 . \quad (2.59)$$

From the Boolean equations (2.58) and (2.59), the equations for the reverse regular expressions are

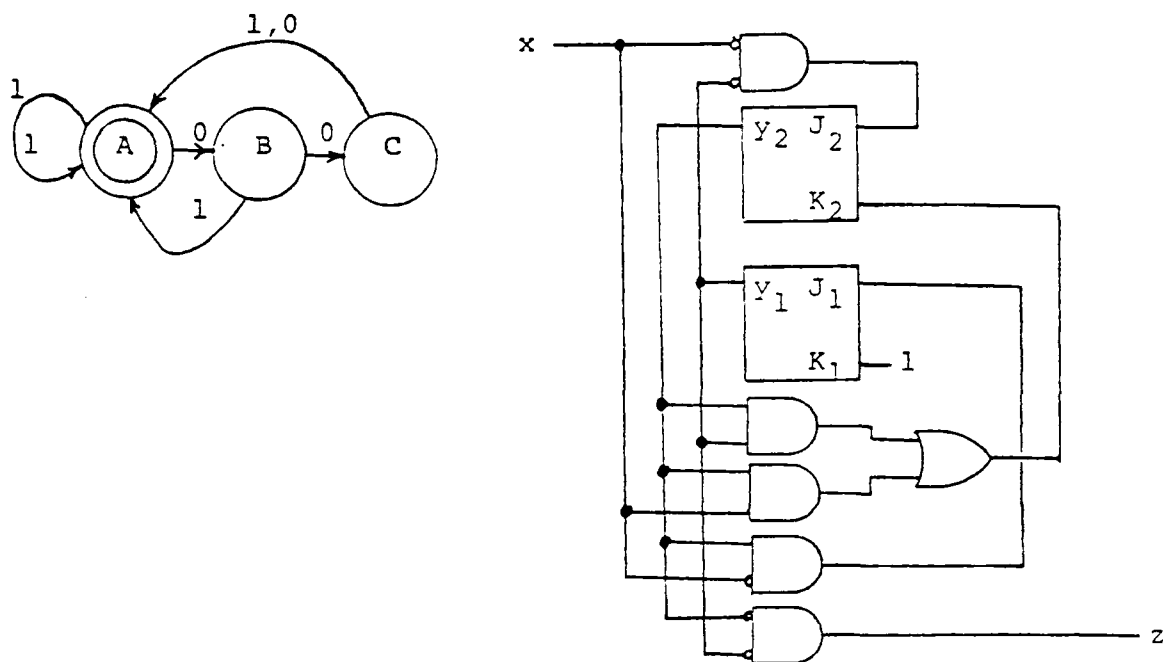


Figure 2.14 State diagram and sequential circuit

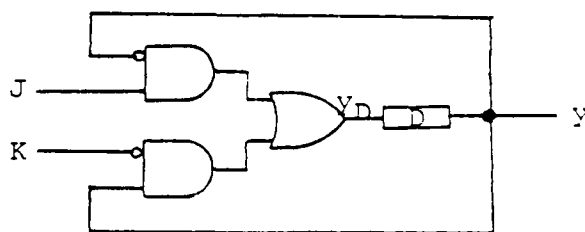


Figure 2.15 J-K flip-flop implemented with delay element

$$\begin{aligned}
r_1 &= \overline{1I} \text{ \& } ir_2 \text{ \& } \overline{ir_1} \\
r_2 &= \overline{1I} \text{ \& } \overline{ir_1} \\
P &= \overline{r_1} \text{ \& } \overline{r_2} \quad (\text{State A}).
\end{aligned}$$

For simplicity the operation A & B will be written

AB. Taking the derivatives we have,

$$\begin{aligned}
D_0 r_1 &= r_2 \overline{r_1} \\
D_0 r_2 &= \overline{r_1} \\
D_1 r_1 &= \phi \\
D_1 r_2 &= \phi.
\end{aligned}$$

Now, $P = \overline{r_1} \overline{r_2}$, thus

$$\begin{aligned}
D_0 P &= \overline{D_0 r_1} \overline{D_0 r_2} = \overline{r_2 \overline{r_1}} r_1 = (\overline{r_2} + r_1) r_1 = r_1 \overline{r_2} + r_1 \\
D_1 P &= \overline{D_1 r_1} \overline{D_1 r_2} = I \\
D_{00} P &= D_0 (D_0 P) = D_0 r_1 \overline{D_0 r_2} + D_0 r_1 = r_2 \overline{r_1} r_1 + r_2 \overline{r_1} = r_2 \overline{r_1} \\
D_{01} P &= \phi \\
D_{000} P &= D_0 r_2 \overline{D_0 r_1} = \overline{r_1} (\overline{r_2} + r_1) = \overline{r_1} \overline{r_2} \\
D_{001} P &= \phi \\
D_{0000} P &= \overline{D_0 r_1} \overline{D_0 r_2} = r_1 \overline{r_2} + r_1 = D_0 P \\
D_{0001} P &= \overline{D_1 r_1} \overline{D_1 r_2} = \overline{\phi} \& \overline{\phi} = I.
\end{aligned}$$

No new derivatives can be found. Therefore, the process is terminated. Now,

$$\begin{aligned}
D_{000} P &= 0D_{0000} P + 1D_{0001} P = 0D_0 P + 1I \\
D_{00} P &= 0D_{000} P + 1D_{001} P = 0D_0 P + 01I \\
D_0 P &= 0D_{00} P + 1D_{01} P = 00D_0 P + 001I.
\end{aligned}$$

By noting that the solution of $X = AX + B$ is $X = A^*B$ we have

$$D_0 P = (000) * 001 I.$$

Now,

$$P = 0D_0P + 1D_1P = 0(000)*001I + 1I.$$

Reversing P we obtain the regular expression for state A .

$$P = 11(00(000)*0+\lambda).$$

However, we note that $(PQ)*P = P(QP)*$ and $\lambda + PP* = P*$. Thus,

$$P = 11(\lambda + 000(000)*) = 11(000)*$$

According to Section 2.4, P is in synchronized form and therefore the steady-state probability of P can be written directly from the regular expression

$$\text{Prob (State A)} = \frac{X}{1-(1-X)^3} = \frac{1}{X^2-3X+3}.$$

Note that the regular expression derived directly from the circuit, by simple manipulations, was put into synchronized form. This property is also supported by the examples to follow. Hence, the method of obtaining regular expressions from sequential circuits is a powerful procedure for obtaining synchronized regular expressions suitable for the derivation of steady-state probabilities.

The following example illustrates how synchronized regular expressions can be found for multiple input devices. From the regular expression, the steady-state probability can be found.

Example 2: Consider a J-K flip-flop shown in Figure 2.15. The input alphabet $A_4 = [a_0, a_1, a_2, a_3]$ is defined as

$$\begin{array}{ll} a_0=0 & (J=0, K=0) \\ a_1=1 & (J=1, K=0) \\ a_2=2 & (J=0, K=1) \\ a_3=3 & (J=1, K=1). \end{array}$$

Now, noting that

$$y_D = J\bar{y} + \bar{K}y ,$$

we can write for the regular expression R of the delay element input, y_D ,

$$R = I(1+3) \ \& \ \bar{R}i + \overline{I(2+3)} \ \& \ Ri$$

where $i=(0+1+2+3)$ and $I=i^*$. Writing the reverse regular expression we obtain

$$r=(1+3)I \ \& \ i\bar{r} + \overline{(2+3)I} \ \& \ ir .$$

As in the previous example we use the concept of the derivative and the relationships between multiple derivatives given by (2.51) and (2.52) to solve the above equation. Therefore, taking the derivatives of r until the derivatives do not repeat we obtain,

$$D_0 r = r$$

$$D_1 r = I \ \& \ \bar{r} + r = r + \bar{r} = I$$

$$D_2 r = \phi$$

$$D_3 r = I \ \& \ \bar{r} = \bar{r}$$

$$D_3 r_0 = \bar{r} = D_3 r$$

$$D_{31} r = \overline{D_1 r} = \phi$$

$$D_{32} r = \overline{D_2 r} = I , \ D_{33} r = r.$$

Now, from (2.51),

$$D_3 r = 0D_{30} r + 1D_{31} r + 2D_{32} r + 3D_{33} r$$

$$D_3 r = 0D_3 r + 2I + 3r$$

$$D_3 r = 0*(2I + 3r).$$

Also,

$$r = 0D_0r + 1D_1r + 2D_2r + 3D_3r$$

$$r = 0r + 1I + \phi + 30*2I + 30*3r$$

$$r = (0 + 30*3)r + (1 + 30*2) I$$

$$r = (0 + 30*3)*(1 + 30*2) I.$$

Reversing the above expression

$$R = I(1 + 20*3) (0 + 30*3)*. \quad (2.60)$$

The above expression is the regular expression of a J-K flip-flop expressed in synchronized form. The steady-state output probability can be found from the synchronized regular expression by noting that

$$P(0) = (1-j)(1-k)$$

$$P(1) = j(1-k)$$

$$P(2) = (1-j)k$$

$$P(3) = jk,$$

where j and k are the J and K input probabilities, respectively. Thus the output probability can be written from (2.60) as

$$P = [j(1-k) + (1-j)k] \frac{1}{1 - (1-j)(1-k)} \frac{1}{1 - [(1-j)(1-k) + \frac{jkjk}{1 - (1-j)(1-k)}}.$$

The expression above can be simplified and the final result is given by

$$P = \frac{j}{j+k}.$$

The above result is consistent with the result obtained through steady-state Markov analyses (Section 2.2).

Note that again the regular expression derived from the circuit was in synchronized form. This property is supported by the next example.

Example 3: Consider a sequential circuit with state diagram given in Figure 2.16. The Boolean excitation functions of the delay elements used to implement the sequential circuit are given by

$$y_1^D = xy_2\bar{y}_1 + \bar{x}y_1 + \bar{y}_2y$$

$$y_2^D = \bar{x}\bar{y}_2 + y_1\bar{y}_2 + \bar{y}_1y_2.$$

State	y_1	y_2
A	0	0
B	0	1
C	1	1
D	1	0

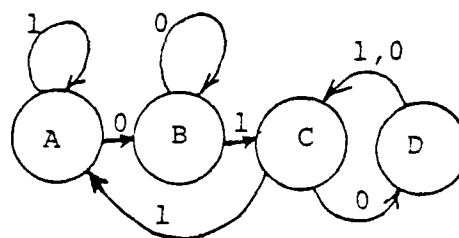


Figure 2.16 State diagram

Proceeding as in the previous examples we derived the regular expression directly from the circuit. The regular expression for state B is thus,

$$R_B = 11 [010(i0)*0]*100*.$$

Noting that $S_B = 1[010(i0)*0]*10$ is the synchronization sequence for this state, we can write

$$S_C = 11[010(i0)*0]*101$$

$$S_D = 1[010(i0)*0]*1010$$

$$S_A = 1[010(i0)*0]*1011.$$

Also, from the state diagram the returning sequences for each state are

$$T_A = 1^*$$

$$T_B = 0^*$$

$$T_C = (0i)^*$$

$$T_D = (i0)^*.$$

Thus,

$$R_A = 11[010(i0)^*10111]^*$$

$$R_B = 11[010(i0)^*0]^*100^*$$

$$R_C = 11[010(i0)^*0]^*101(0i)^*$$

$$R_D = 11[010(i0)^*0]^*1010(i0)^*.$$

We can obtain the steady-state state probabilities from the above synchronized regular expressions. For state B we have

$$P_B = p \frac{1}{1 - \frac{p(1-p)^3}{1 - 1(1-p)}} p(1-p) \frac{1}{1 - (1-p)} = \frac{1-p}{p^2 - 3p + 3},$$

where p is the probability that the input is a one. Again the result is consistent with the result obtained through Markov steady-state analysis.

2.7 Comparison of Methods

In this chapter methods for obtaining output probabilities as a function of input probabilities for sequential circuits were investigated. When the state diagram of the sequential circuit is available, we showed that using Markov steady-state analysis a system of equations for the state probabilities can be directly obtained from the

state diagram. Solution of the system of equations, however, for complex state diagrams is difficult and not straightforward. Situations exist, however, where general expressions for state probabilities of iterative type sequential circuits can be derived as was shown for a general n-bit programmable counter. A more efficient method for obtaining state and output probabilities was to consider the synchronized regular expression from which the probability equation can be directly obtained. Synchronized regular expressions, however, are difficult to obtain from the state diagram of sequential circuits. This is because the synchronizing sequences of the circuit must be derived and the resulting regular expression must be unique.

The above methods, however, are independent of circuit implementation and are derived from the state diagram or verbal description of the sequential circuit operation. Therefore, we examined methods in which the state and output probabilities are derived directly from the circuit. The first method involved taking advantage of the iterative structure of some sequential circuits. Problems were encountered, however, due to the statistical dependence between internal signal lines and the method was limited to certain iterative structures. A more general method for deriving state and output probabilities is to derive the synchronized regular expression directly from the circuit from which the probabilities can be found. The method has the potential of being automated; however, more research is required.

The advantage of the methods for obtaining state and output probabilities directly from the sequential circuit is that faults can be conveniently inserted into the circuit and the corresponding change in output probability can be found. However, when considering modular sequential circuits, where faults are limited to single stuck-at faults at the inputs to the module, the Markov steady-state analysis or the regular expression description of the sequential circuit can be used since the internal structure of the sequential circuit is unimportant in this case.

3 ANALYSIS OF A FOUR-STATE SEQUENTIAL CIRCUIT

We can now apply the methods of Chapter 2 to the analysis of a four-state sequential circuit. In this analysis we will demonstrate the effectiveness of the BIT strategy, outlined in Chapter 1, for testing the circuit. First, the state probabilities are derived in terms of the input probability distribution. Secondly, faults are inserted into the circuit and the corresponding changes in state diagram and state probabilities are tabulated. Next, using the tabulated faulty circuit state probabilities, the fault density function for the given fault set is obtained. From this function, the escape probability is calculated for each state and plotted as a function of the input probability distribution. Finally, a discussion of the results and the general behavior of sequential circuits under faults is presented.

A simple single input four-state sequential circuit will be analyzed in order to demonstrate the effectiveness of the referenceless random testing of the circuit. The machine in question is described by the state diagram given in Figure 3.1. We will be concerned with the monitoring of the statistics of the states of the machine under random inputs. Therefore, the outputs of the machine are simply state decoders.

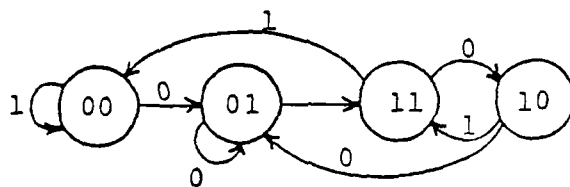


Figure 3.1 State diagram

An implementation of the machine using J-K flip-flops is shown in Figure 3.2. Table 3.1 lists the Boolean relationships of the excitation circuit for faulty circuits.

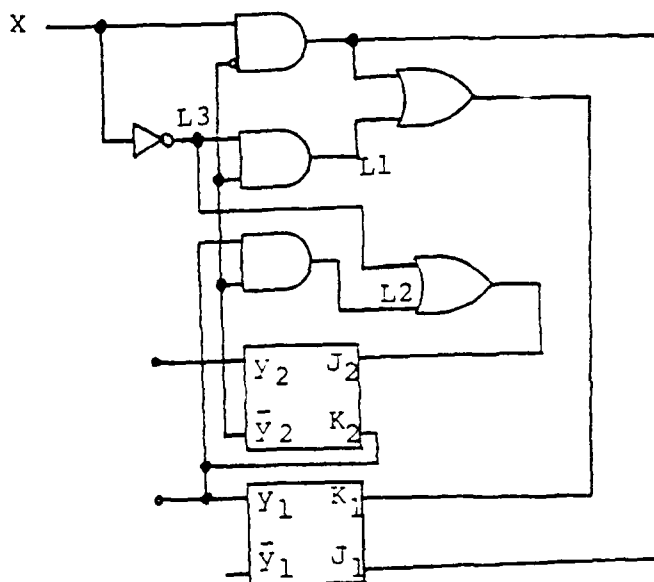


Figure 3.2 Circuit diagrams

Table 3.1 Enumeration of faults in the network of Figure 3.2

Fault/State Eqns.	State Diagram	State Problem
① $J_1 \text{ s-a-0}$ $K_1 = \bar{n}\bar{y}_2$ $J_2 = \bar{n} + y_1\bar{y}_2$ $K_2 = y_1$		$P_A \rightarrow 0$ $P_B \rightarrow 1$ $P_C \rightarrow 0$ $P_D \rightarrow 0$
② $J_1 \text{ s-a-1}$ $K_1 = \bar{n}\bar{y}_2$ $J_2 = \bar{n} + \bar{y}_1 y_2$ $K_2 = y_1$		$P_A = \frac{1}{2+2p}$ $P_B = \frac{p}{2+2p}$ $P_C = \frac{1}{2+2p}$ $P_D = \frac{p}{2+2p}$
③ $K_1 \text{ s-a-0}$ $J_1 = ny_2$ $J_2 = \bar{n} + y_1\bar{y}_2$ $K_2 = y_1$		$P_A \rightarrow 0$ $P_B \rightarrow 0$ $P_C \rightarrow 0.5$ $P_D \rightarrow 0.5$
④ $J_2 \text{ s-a-0}$ $J_1 = ny_2$ $K_1 = \bar{n}\bar{y}_2 + ny_2$ $K_2 = y_1$		$P_A \rightarrow 1$ $P_B, P_C, P_D \rightarrow 0$
⑤ $J_2 \text{ s-a-1}$ $J_1 = Ky_2$ $K_1 = \bar{n}\bar{y}_2 + ny_2$ $K_2 = y_1$		$P_A = p^2 / (1+p+p^2)$ $P_B = (1-p+p^2) / (1+p+p^2)$ $P_C = p / (1+p+p^2)$ $P_D = (p-p^2) / (1+p+p^2)$
⑥ $L_1 \text{ s-a-0}$ $J_1 = ny_2$ $K_1 = \bar{n}y_2$ $K_2 = y_1$		$P_A = p/p^2 - 3p - 3$ $P_B = (1-p)/p^2 - p + 3$ $P_C = (1-p)/p^2 - 3p + 3$ $P_D = (1-p)^2/p^2 - 3p - 3$

Table 3.1 (continued)

L_2 s-a-0 ⑦ $J_1 = ny_2$ $K_1 = \bar{n}y_2 + ny_2$ $K_2 = y_1$		$P_A = p^2 / (1+p-p^2)$ $P_B = \frac{1-p}{1+p-p^2}$ $P_C = \frac{p^1-p^2}{1+p-p^2}$ $P_D = \frac{p-p^2}{1+p-p^2}$
L_3 s-a-0 ⑧ $J_1 = ny_2$ $K_1 = ny_2$ $J_2 = y_1 y_2$ $K_2 = y_1$		$P_A = 1$ $P_B, P_C, P_D = 0$
L_3 s-a-1, L_1 s-a-1 ⑨ $J_1 = ny_2$ $K_1 = \bar{y}_2 + ny_2$ $J_2 = 1$ $K_2 = y_1$		$P_A = \frac{p}{1+2p}$ $P_B = \frac{1}{1+2p}$ $P_C = \frac{p}{1+2p}$ $P_D = 0$

*p is the input vector probability

From the state diagram, we can write down the system of equations for the steady-state probabilities:

$$P_A = pP_A + pP_C \quad (3.1a)$$

$$P_B = (1-p)P_A + (1-p)P_B + (1-p)P_D \quad (3.1b)$$

$$P_C = pP_B + pP_D \quad (3.1c)$$

$$P_D = (1-p)P_C \quad (3.1d)$$

$$P_A + P_B + P_C + P_D = 1 \quad (3.1e)$$

Solving these equations for the state probabilities we obtain (3.2 a-d).

$$P_A = p^2 \quad (3.2a)$$

$$P_B = (1-p)(1-p+p^3) \quad (3.2b)$$

$$P_C = p(1-p) \quad (3.2c)$$

$$P_D = (1-p)^2 p \quad (3.2d)$$

We will now examine how faults modeled as simple stuck-at-zero or -one faults change and modify the state diagram, and hence the steady state probabilities of the circuit. The results corresponding to various faulty configurations are tabulated in Table 3.1. At this point in the development the objective is to exemplify; for that reason a very straightforward fault model is used. A more realistic and thorough fault model is developed in Chapter 4.

The manner in which the state diagram is modified differs for each faulty condition. We observe from Table 3.1 that some faults (J_1 s-a-0, J_2 s-a-0, and L_3 s-a-0) cause some states to be absorbing. In some cases (L_3 s-a-1, K_1 s-a-0), the number of states of the machine is reduced. However, some faults cause only the transition between states to be effected (J_1 s-a-0, J_2 s-a-1, L_1 s-a-0, L_2 s-a-0). This category of faults may be the most difficult to detect.

Statistical Testing: Considering the fault set of Table 3.1, we next examine the efficiency of referenceless random testing in detecting these faults. The test is performed by applying a number of random inputs to the circuit and counting the number of occurrences of a logical one at the output. This count is then compared to the probability of the fault-free output and the circuit passes the test if its count, Z_C , satisfied the condition

$\bar{z}_j - \epsilon \leq \frac{z_c}{N} \leq z_j + \epsilon$ where z_j is the fault-free probability and ϵ is the test stringency, and N is the test length. In order to determine the efficiency of the test we must calculate the escape probability. The probability of escape can be obtained by (3.3) also known in statistics as the error of the second kind.

$$P_{\text{esc}} = \sum_{K=N(z_j-\epsilon)}^{N(z_j+\epsilon)} \int_0^1 \binom{N}{K} z^K (1-z)^{N-K} \phi(z) dz \quad (3.3)$$

In the above equation $\phi(z)$ is the density of faulty circuits with output probability z and N is the test length. Since the fault-free output probability z_j is a function of the input probability x , the escape probability can be obtained as a function of input probability. We must, however, first obtain the fault density function $\phi(z)$.

Since $\phi(z)$ is a probability function we must statistically model the faults. For convenience we will assume that all faults in the fault set of Table 3.1 are equiprobable. The fault density function is thus

$$\phi(z) = \frac{1}{M} \sum_{i=1}^M \delta[z - f_i(x)] \quad (3.4)$$

where M is the number of faults, $\delta(z)$ is the impulse function defined by (3.5) and $f_i(x)$ is the function relating the output to input probability for faulty circuit i .

$$\delta(z) = \begin{cases} 1 & \text{if } z = 0 \\ 0 & \text{if } z \neq 0 \end{cases} \quad (3.5)$$

Note that we can define the output z to be such that it is a logical one on the occurrence of a specified state. Moreover, we will be concerned with the monitoring of the states of the machine.

A three-dimensional plot of the number of faulty circuits with output probability z_f such that $z - \epsilon \leq z_f \leq z + \epsilon$ with fixed input probability is shown in Figure 3.3. This plot is for state A with $\epsilon = 0.05$.

Results and test efficiency: From (3.3) and knowledge of the functions $f_i(x)$ obtained from Table 3.1 we can calculate the probability of escape for a given experiment length and test stringency. To simplify the calculations we make use of the approximation

$$\binom{N}{K} p^K (1-p)^{N-K} \approx \frac{1}{\sqrt{2NP(1-P)}} e^{-(K-NP)^2/2NP(1-P)} \quad (3.6)$$

which is valid when N is large, and $NP(1-P) \gg 1$. The probability of escape of state B versus the input probability for three experiments was obtained and is shown in Figure 3.4. We observe that by increasing the test length the probability of escape is not decreased by an appreciable amount. Tightening the test stringency, however, decreases the probability of escape significantly. In addition, we observe that the best region to test the circuit, in an off-line situation, is to set the input probability in the region $0.4 \leq x \leq 0.5$.

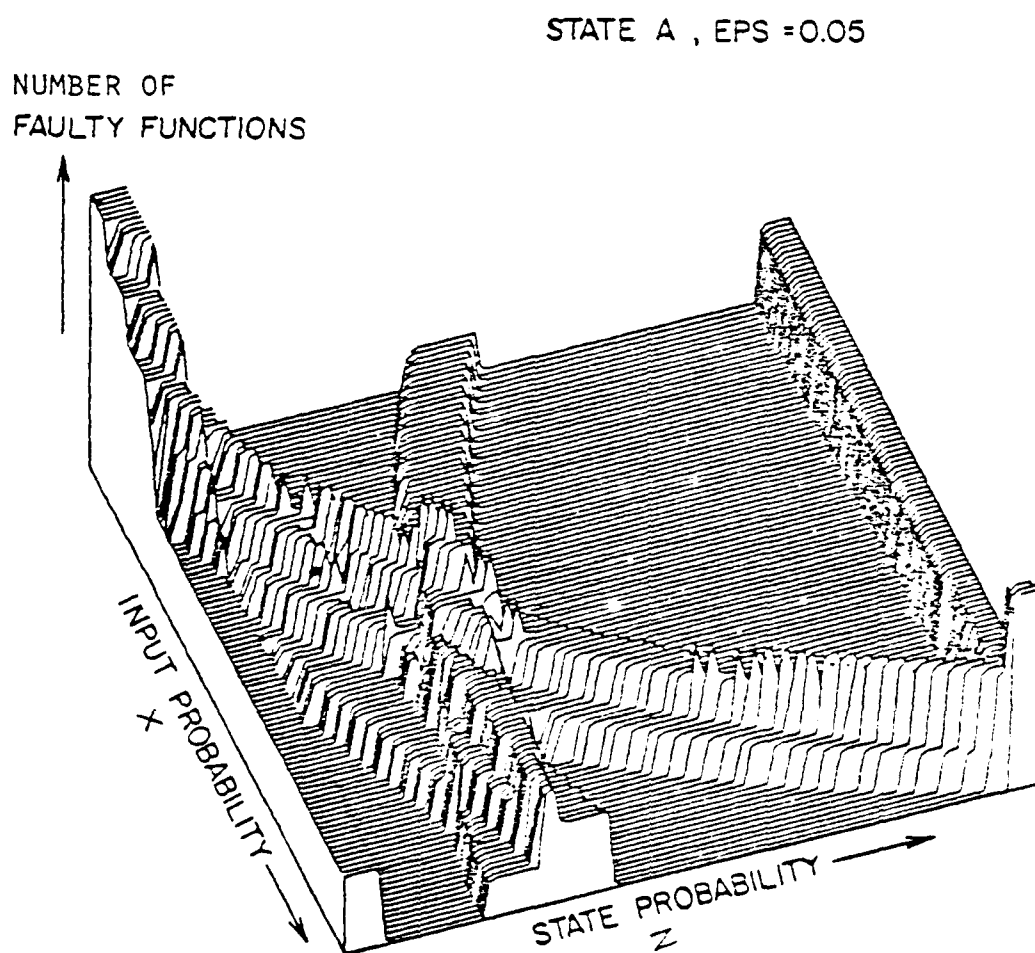


Figure 3.3 Number of faulty circuits with $z - \epsilon \leq z_c \leq z + \epsilon$

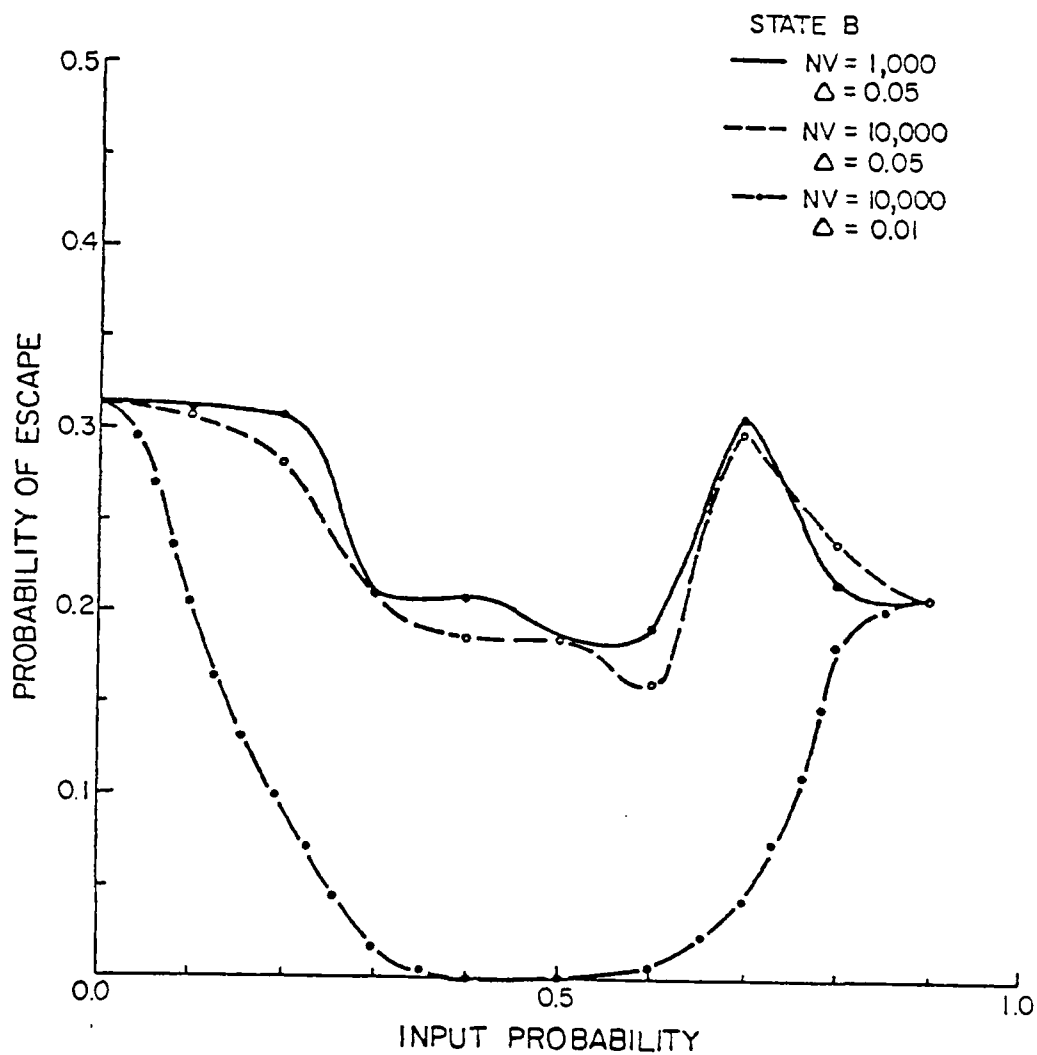


Figure 3.4 Probability of escape, state B

We must keep in mind that the probability of false alarm as an upper bound is related to the test length and test stringency, by the relationship [8],

$$\text{Prob (FA)} = \text{Erfc} (\epsilon\sqrt{2N}) \quad (3.7)$$

which is plotted in Figure 3.5. We see that increasing the test length makes the test more efficient by reducing both the false alarm rate and escape probability. Tightening the test stringency, however, increases the false alarm rate although it significantly decreases the escape rate. Therefore, a compromise must be made by the test designer regarding the selection of the test stringency.

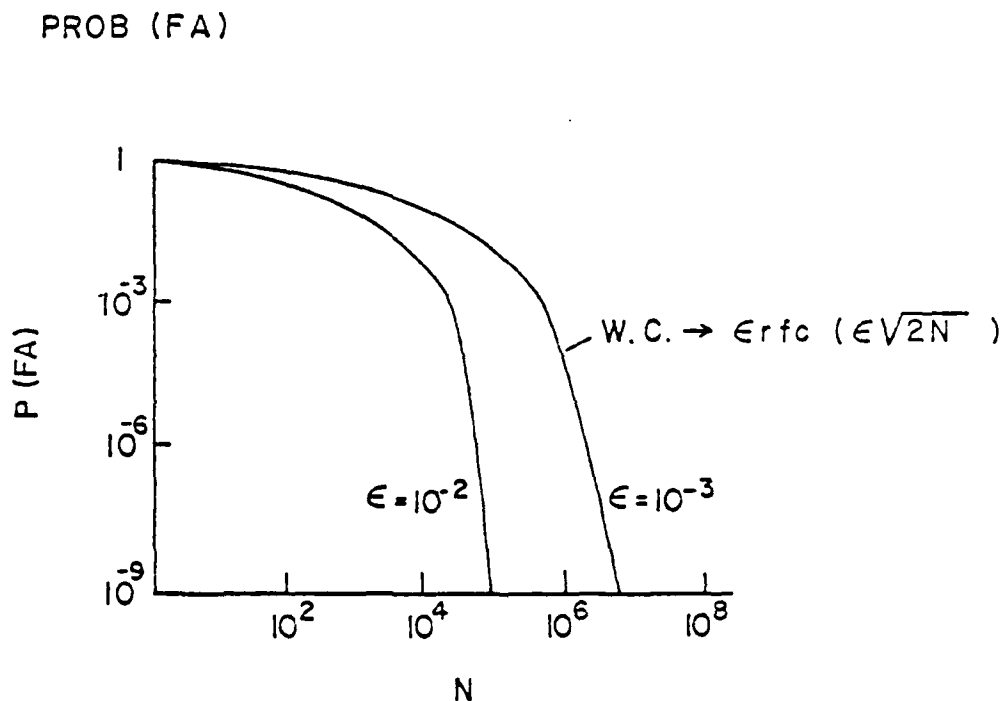


Figure 3.5 Probability of false alarm

To determine which states to monitor in order to efficiently fail faulty circuits over a wide input probability range, we examine the probability of escape vs. the input probability for the four states of the machine shown in Figure 3.6. These results are for experiments with a test length $N=10,000$ and test stringency $\varepsilon = 0.01$.

From Figures 3.6a and b, we see that if both states A and B are monitored the escape probability will be quite low over the range of input probabilities $0.3 \leq x \leq 0.6$. While for $x=0.5$ the escape probability of state A increases to 0.1, the escape probability of state B remains quite low. We may say that these two states "compensate" each other. Furthermore, no gain is achieved by monitoring states C and D if states A and B are monitored.

Conclusions: These results show that in order to determine the best test procedure, i.e., the determination of which states to monitor and the specification of test length and test stringency, an accurate method must be developed for the calculation of the probability of escape. Especially in an on-line environment, where the input statistics are uncontrollable, accurate knowledge of the probability of escape as a function of the input probability allows us to determine the efficiency of the testing procedure for the given on-line input statistic. Moreover, these results show that statistical fault monitoring is quite efficient in detecting possible faults in the four

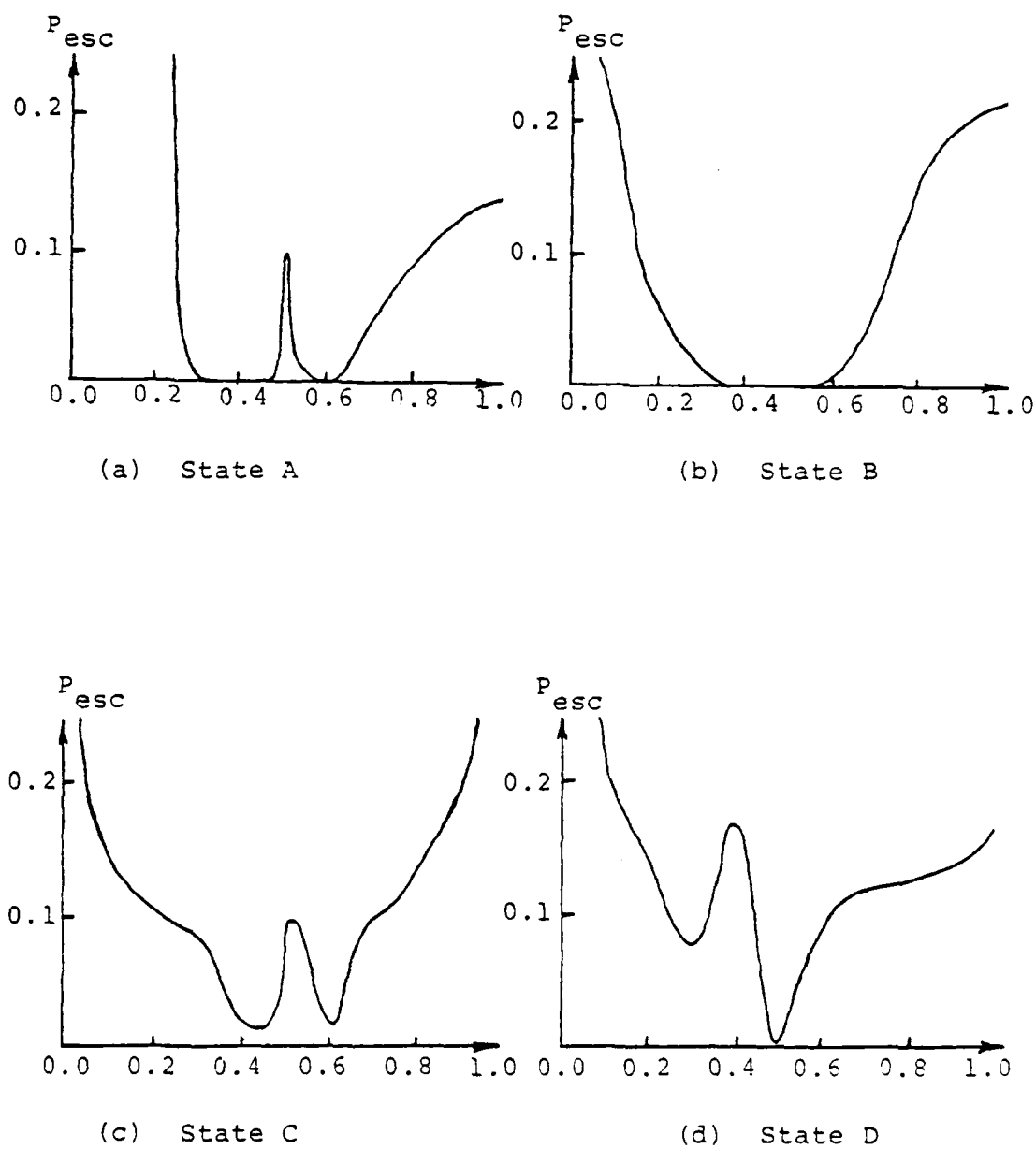


Figure 3.6 Probability escape vs. input probability ($N=10,000$, $\epsilon = 0.01$)

state machine examined. However, the derivation of a measure for the efficiency of the test proved to be quite vigorous. It was observed that faults may change the state diagram of the machine quite arbitrarily. Furthermore, relating the output statistics to the input statistics for each faulty circuit is difficult. This is because a state diagram must be constructed from each faulty circuit — a lengthy process — and from the state diagram a system of equations must be derived from which the steady-state probabilities are obtained. As was described in section 2.4.2 the derivation of the regular expression from a sequential circuit and the consequent calculation of the steady-state probability from the regular expression may provide an efficient method for obtaining the state probabilities of faulty circuits. Nevertheless the procedure is still lengthy. In the next chapter, therefore, we will deal with a statistical fault model that reduces the computations and yet provides good fault coverage for sequential integrated circuits.

4 STATISTICAL PIN-FAULT MODEL

4.1 Introduction

Primary to the evaluation of the probability of escape and therefore to the determination of effectiveness of a given test is the derivation of the fault density function. Specifically the probability of escape is related to the fault density function $\phi(z)$ according to the expression [8]

Prob (Pass Test/faulty) =

$$\sum_{k=N(Z_j-\epsilon)}^{N(Z_j+\epsilon)} \int_0^1 \binom{N}{K} z^K (1-z)^{N-K} \phi(z) dz$$

where N and ϵ are the test length and stringency respectively. The function $\phi(z)$ which depends on the input statistics is derived from a given circuit under a specified fault model. In Section 4.2 we develop a statistical fault model under the pin-fault assumption proposed by Ketelsen [6]. Section 4.3 shows how the fault density function for a module is obtained from the statistical fault model and knowledge of the output probability expression. An algorithm for computing the fault density function is given in Section 4.4. This algorithm is used to compute the escape probabilities of two integrated sequential circuits treated in Chapter 5.

4.2 Statistical Pin-Fault Model

An appropriate fault model for fielded digital electronic equipment is the pin-fault model proposed and partly

justified by Ketelson [6]. This model incorporates faults occurring due to, for example, "lead bond" failures, failures of input or output transistors, or failures due to improper connection of the package pins to the silicon chip. According to this model, the input and output pins of integrated circuits experience stuck-at-one or stuck-at-zero faults.

In this section a statistical pin-fault model will be developed for digital synchronous sequential circuits. The abstract model of a synchronous sequential circuit is shown in Figure 4.1. In this mode inputs occur at discrete intervals of time and each application of inputs results in, at most, one state transition. The delay elements are implemented using suitable flip-flops (master-slave or edge triggered) to insure proper synchronous operation [6].

We will examine the model in order to determine in an abstract way the effect of various faults on the statistics of the outputs. Referring to Figure 4.1, stuck at faults at the outputs, y_j , of the delay elements cause the number of states to decrease. This causes significant deviation of the faulty circuit statistic from the fault-free statistic, particularly if the outputs are the internal states of the machine. This deviation of output statistics due to faults on the delay element outputs is thus immediately detectable. The same arguments can be stated for faults on the D-type delay element input lines. If the

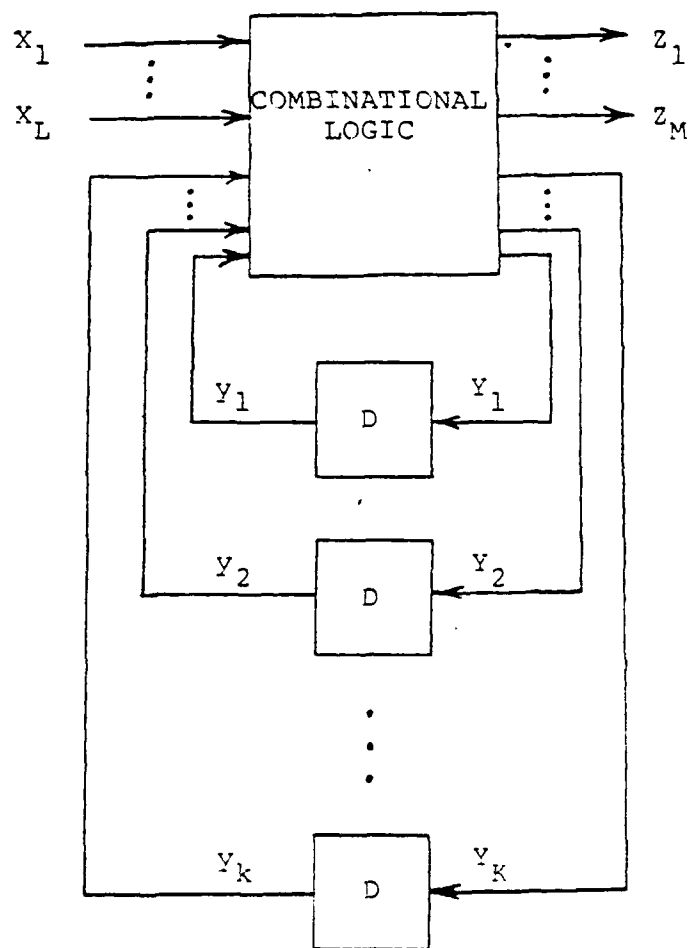


Figure 4.1 Model of synchronous sequential circuit

delay elements were implemented with JK flip-flops the nature of the effects of faults on the input lines of the delay elements is more complicated. Faults on the inputs of set-reset flip-flops, however, cause the outputs to be stuck-at-one or stuck-at-zero and thus are detectible as in the case of D-type flip-flops.

Unlike the faults mentioned above, faults occurring on the input lines, X_k , to the sequential circuit may cause minor changes in the output statistics. For example, a failure may change the transition from state Q_i to Q_j under input A_k ($0 \leq A_k \leq 2^L-1$). According to Losq [8], "these failure will be far more difficult to detect because their effect on the system operation is somewhat limited (in a statistical sense)." Faults occurring in the combinational logic also may cause minor changes in output statistics.

Since, according to the above discussion, faults occurring at the input pins to the sequential circuit dominate other faults, we propose a fault model that considers faults to occur only at the input pins of the sequential circuit. Hence, if the test strategy is capable of detecting faults at the input pins, faults at the delay elements and output pins are also detected. This fault model is known as the pin-package fault model. We note that we exclude faults occurring in the combinational logic since we are dealing with fielded integrated circuits and, therefore, faults occurring in the combinational logic

have a very low probability of occurrence compared to faults at the input pins (i.e., for a silicon chip the probability that a single gate function fails is very low for fielded integrated circuits). However, an advantage of statistical testing of the outputs is that, although knowledge of the effect of faults in the combinational circuit on the output is difficult to analyze, it can be shown that most faults cause deviations from the fault-free statistics and are thus detectible (see Chapter 3). The exclusion of the analyses of these faults in the fault model greatly simplifies the analyses of the problem and makes the systematic analyses of the test strategy possible while at the same time a large percentage of the high probability faults are taken into account.

In our discussion we will be concerned with integrated circuit implementations of the model of Figure 4.1. In this case we have access only to the input and output pins of the circuit. Thus, in the pin-fault model faults in the form of stuck-at-one or stuck-at-zero faults occur only on the input and output pins. We place no restriction on the number of pins that can be at fault. As was discussed above, this model provides good fault coverage.

The total number of faulty input combinations assuming stuck-at-one and stuck-at-zero faults at the pins of an input circuit is

$$N_f = 3^L - 1. \quad (4.1)$$

If we let k denote the number of pins at fault, then the number of possible combinations of k stuck-at faults is

$$n_f = 2^k \binom{L}{K}. \quad (4.2)$$

We assign a probability P_k to the event that k pins are at fault simultaneously. We have then, from (4.2),

$$\sum_{k=1}^L 2^k \binom{L}{K} P_k = 1. \quad (4.3)$$

It is reasonable to assume that the probability of occurrence of multiple pin faults is a decreasing function of the number of faulty pins. In other words, P_k decreases as k is increased. In this study we assume that

$$P_k = e^{-\alpha K}. \quad (4.4)$$

Thus, in the statistical fault model considered, the probability of faults at the input pins decreases exponentially as the number of simultaneous faulty pins increases. The exponent coefficient α corresponding to the number of input pins L is listed in Table 4.1.

Let the variable f be an integer defined in the interval $1 \leq f \leq 3^L - 1$. Then each integer f defines a specific fault configuration with a corresponding number K simultaneous pin faults. We thus define the function $g(f)$ as the probability of the occurrence of the faulty configuration f .

This function will be used in the derivation of the fault density function in the next section. The variable

Table 4.1 α parameter (see program ALPHA in appendix)

Number of input pins	α
1	0.6931472
2	1.574520
3	2.040524
4	2.358062
5	2.598983
6	2.793101
7	2.955652
8	3.095474
9	3.218129
10	3.32738
11	3.425883
12	3.515547
13	3.597834
14	3.673863
15	3.744524
16	3.810523

f can be ordered such that the number of faulty pins specified by faulty configuration f increases as f increases. The function $g(f)$ plotted against such an ordering of f is shown in Figure 4.3.

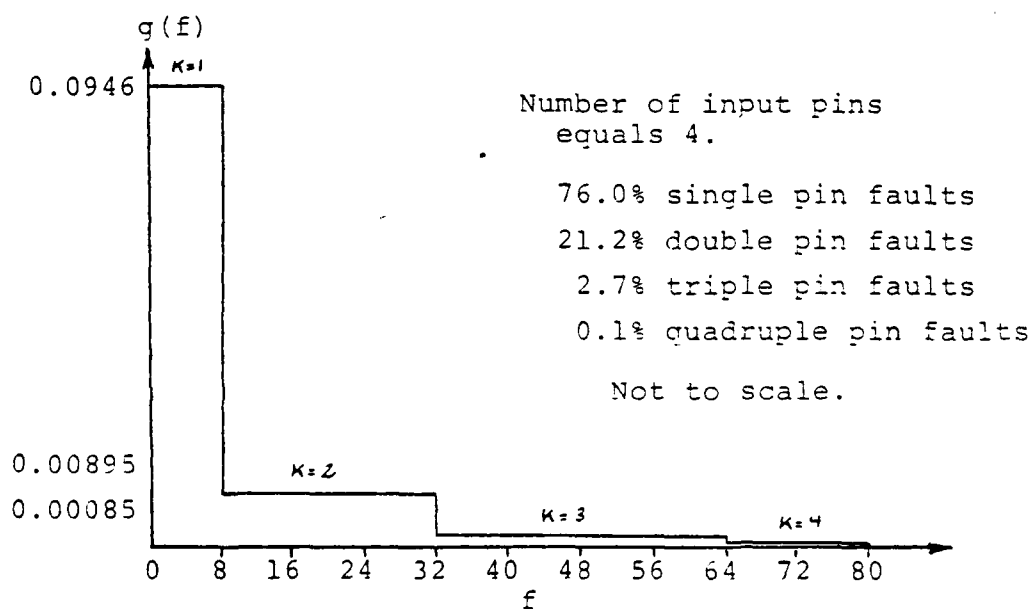


Figure 4.3 Pin-fault probability distribution

4.3 Derivation of $\phi(z)$

In this section the fault density function will be derived for a synchronous sequential integrated circuit using the statistical pin fault model. Let the function relating the output probabilities to the input probabilities be $Z=T(x_1, x_2, \dots, x_L)$. Under the pin fault assumption only stuck-at-zero or -one faults occur at the input pins. Thus, when input line x_i , $i=1, 2, \dots$ is at fault the faulty output statistic is either $Z_f=T(x_1, x_2, \dots, x_{i-1}, 0, \dots, x_L)$ or $Z_f=T(x_1, x_2, \dots, x_{i-1}, 1, 0, \dots, x_L)$ depending on whether the pin is stuck-at-zero or stuck-at-one.

Now, as before, let the integer f be in the region $1 \leq f \leq 3^L-1$, then each integer f defines a faulty configuration and, as will be shown in the next section, the corresponding faulty output statistic, Z_f , as a function of the integer f assuming fixed input statistics can be expressed as

$$Z_f = T(f)$$

where $T(f)$ is related to $T(x_1, x_2, \dots, x_L)$ the fault-free transfer function.

The fault density function $\phi(z)$ is the density of faulty circuits that have output statistic z . We can thus arrive at an expression for $\phi(z)$ in terms of the functions $Z_f=T(f)$ and $g(f)$. This expression is given in (4.10).

If we define

$$\delta[z-T(f)] = \begin{cases} 1.0 & \text{if } z=T(f) \text{ i.e., } z=z_f \\ 0.0 & \text{if } z \neq T(f) \end{cases} \quad (4.9)$$

then

$$\phi(z) = \sum_{f=1}^{3L-1} \delta[z-T(f)]g(f) \quad (4.10)$$

As a check to the above derivation, we note that $\int_0^1 \phi(z)dz=1$. Applying this relation to (4.10) we have

$$\begin{aligned} \int_0^1 \phi(z)dz &= \int_0^1 \sum_{f=1}^{3L-1} \delta[z-T(f)]g(f)dz \\ &= \sum_{f=1}^{3L-1} g(f) \int_0^1 \delta[z-T(f)]dz = \sum_{f=1}^{3L-1} g(f)=1. \end{aligned} \quad (4.11)$$

The following section gives an algorithm for computing $\phi(z)$, . . ., evaluating (4.10) on a digital computer. This procedure is then used to accurately calculate the escape probability for digital sequential integrated circuits in a defined test.

4.4 Algorithm for the Calculation of $\phi(z)$

The mathematical discussion in Section 4.3 suggests an algorithm for the computation of the fault density function. We shall attempt at first to develop a method for obtaining a fault configuration (ordered in increasing number of faulty pins).

The total number of faulty configurations is $N=3^{np}-1$ where np is the number of input pins. We define the integer

f in the region $0 \leq f \leq N$. Expanding F in powers of 3 we obtain

$$f = \alpha_1 + 3\alpha_2 + 3^2\alpha_3 + \dots + 3^{np-1}\alpha_{np} \quad (4.12)$$

where $\alpha_i = 0, 1$, or 2 . Thus, each integer f is defined by a set $[\alpha_1, \alpha_2, \dots, \alpha_{np}]$. We shall define $\alpha_i = 2$ to mean that pin i is fault-free and $\alpha_i = 0$ or 1 to mean pin i is stuck-at-zero or -one, respectively. Thus, if $np=4$, the set $[2102]$ representing $f=2+3+0+27 \times 2=59$ means pin numbers one and four are fault-free, pin number two is stuck-at-one and pin number three is stuck-at-zero.

We can obtain the faulty output statistic Z_f corresponding to a faulty configuration $[\alpha_1, \alpha_2, \dots, \alpha_{np}]$ by defining coefficients β_i according to (4.13) and (4.14) and relating the coefficients β_i to α_i .

$$Z_f = T(\beta_1 x_1, \beta_2 x_2, \dots, \beta_i x_i, \dots, \beta_{np} x_{np}) \quad (4.13)$$

$$\beta_i x_i = \begin{cases} x_i & \text{Pin } i \text{ is fault free } (\alpha_i = 2) \\ 0 & \text{Pin } i \text{ is s-a-0 } (\alpha_i = 0) \\ 1 & \text{Pin } i \text{ is s-a-1 } (\alpha_i = 1) \end{cases} \quad (4.14)$$

From (4.14) we observe that

$$\beta_i = \begin{cases} 0 & \alpha_i = 0 \\ 1/x_i & \alpha_i = 1 \\ 1 & \alpha_i = 2 \end{cases} \quad (4.15)$$

We thus obtain a relationship between β_i and α_i which can be expressed as

$$\beta_i = \frac{\alpha_i}{2} [1 + (\alpha_{i-2}) \alpha_i] \quad (4.16)$$

which satisfies (4.15) if $\lambda_i = \frac{x_i - 2}{x_i}$. Substituting for λ_i , we obtain the equation

$$\beta_i = \frac{\alpha_i}{2} \left[1 + (\alpha_i - 2) \frac{x_i - 2}{x_i} \right] \quad (4.17)$$

relating β_i to α_i .

We are now in a position to present the algorithm for computing $\phi(z)$. The first step is to generate the coefficients α_i . Furthermore, these coefficients must be generated in the order of increasing number of faulty pins. For each set of coefficients there is a corresponding probability P_k defined by the relation $P_k = e^{-\alpha k}$ where k is the number of faulty pins in the faulty configuration defined by the coefficients. Thus, as the coefficients are generated, the probability density function $q(f)$ of faulty configurations f is also generated. From (4.17) the coefficients β_i are derived. Using these coefficients, the output probability Z_f corresponding to the faulty configuration f is obtained through (4.13). Finally, as the functions $g(f)$ and $Z_f = T(f)$ are obtained through this procedure, the fault density function is calculated using relationship (4.10).

The above algorithm has been implemented in FORTRAN and is listed in Appendix A.

We have developed a fault model which is computationally feasible and which has a reasonable coverage of high probability faults. In addition, we have shown how the fault

density function $\phi(z)$ can be derived using this model. The final step of using ϕ to compute the probability of escape is treated in the next chapter.

5 COMPUTATIONAL RESULTS AND SIMULATION

5.1 Introduction

In this chapter the probability of escape is calculated and results are presented for two synchronous sequential integrated circuits. The computations are done using the algorithm described in the previous chapter. The results are verified by computer simulations of the devices under various faulty configurations. In the simulations, pseudo-random inputs are applied and the statistics of the monitored outputs of the device are collected. If the output statistics are within the specified range of the fault-free statistics the devices pass the test. Thus, the criterion for passing a simulated test can be summarized by

$$N (Z_i - \epsilon) \leq Z_c \leq N(Z_i + \epsilon)$$

where Z_c is the number of occurrences of a one at the output and N , ϵ , and z_i are the number of simulations (test length), test stringency, and fault-free statistic, respectively. It may be required that all outputs satisfy the above condition.

5.2 Escape Probability of a Two-Bit Right-Shift Parallel-Load Shift Register

The logic diagram of a two-bit right-shift, parallel-load shift register is shown in Figure 5.1. The relationships between output and input probabilities (steady-state) are given in Table 5.1. Using these relationships and the

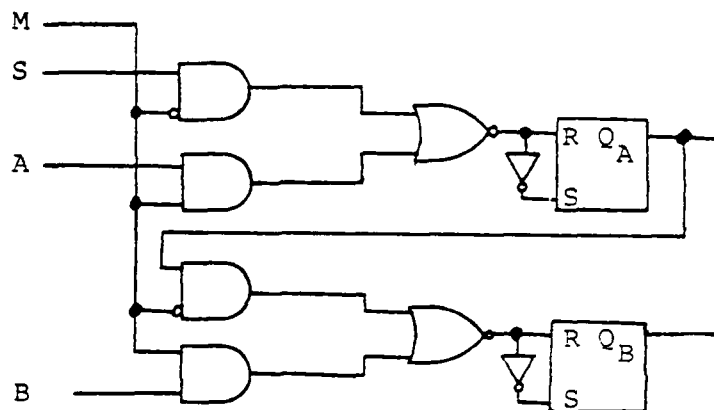


Figure 5.1 Logic diagram of shift register

Table 5.1 State probabilities

State ($Q_B Q_A$)	State Probability Expression*
A (00)	$\bar{s}\bar{m} - s\bar{s}\bar{m}^2 - \bar{s}am\bar{m} + \bar{b}\bar{a}m$
B (01)	$s\bar{m} - s^2\bar{m}^2 - sam\bar{m} + \bar{b}am$
C (11)	$s^2\bar{m}^2 + asm\bar{m} + mab$
D (10)	$\bar{m}^2s\bar{s} + \bar{m}m\bar{s}a + b\bar{a}m$

statistical pin-fault model, the probability of escape was obtained for each output. These results are based on specific input probabilities given in Table 5.2 and a test length $N=10,000$ and test stringency $\epsilon=0.01$. In obtaining the results, the statistics of one pin was varied between zero and one while the statistics of the remaining pins were held constant. Figures 5.2 through 5.5 show plots of the results for each state.

Table 5.2 Input probabilities

Input Pin	Probability
Mode (M)	0.8
Serial in (S)	0.5
Parallel in (A)	0.3
Parallel in (B)	0.6

These plots show that very good testing regions exist where the probability of escape is extremely low. For instance, with the input statistics given in Table 5.2 ($b=0.4$) the probability of escape by monitoring state B is $P_{esc}=2.5 \times 10^{-11}$. These results suggest that statistical methods are useful in testing this type of circuit. In fact, if the input statistics are controllable, using the plots obtained tests resulting in extremely low escape probabilities can be designed ($P_{esc}=5.3 \times 10^{-17}$ for $m=0.8$, $s=0.5$, $a=0.3$, and $b=0.2$ by monitoring state A, $N=10,000$, $\epsilon=0.01$). However, in an on-line environment where prior knowledge of the input statistics exists — input statistics are uncontrollable — using the computational procedures developed, the escape probability and therefore the test efficiency can be evaluated for a given experiment. The plots obtained for the probability of escape show that in an on-line situation if the inputs exhibit random behavior, regions exist for which statistical testing of circuits is quite attractive (i.e., low probability of escape).

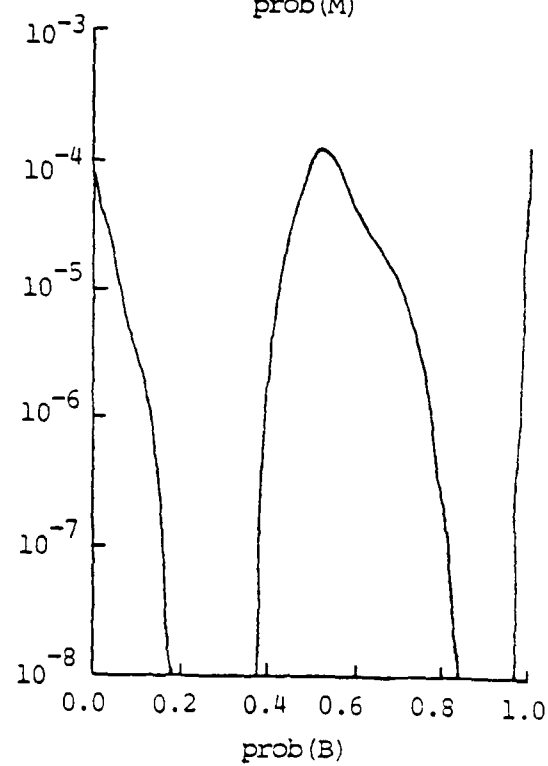
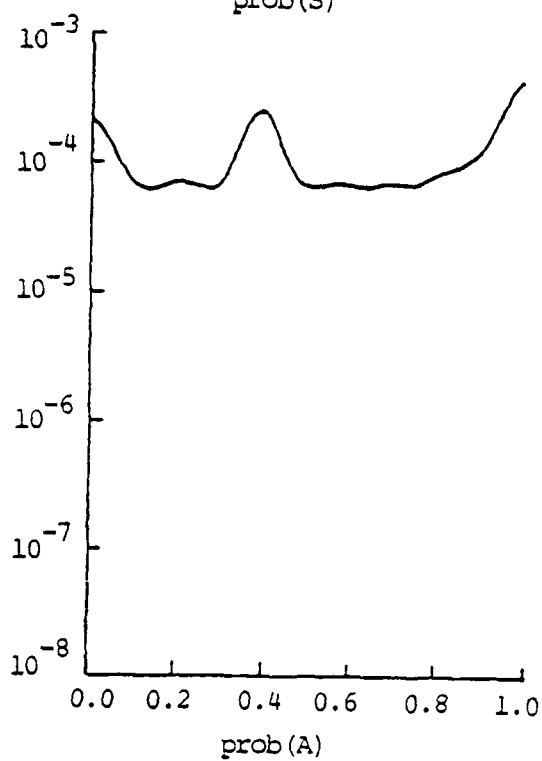
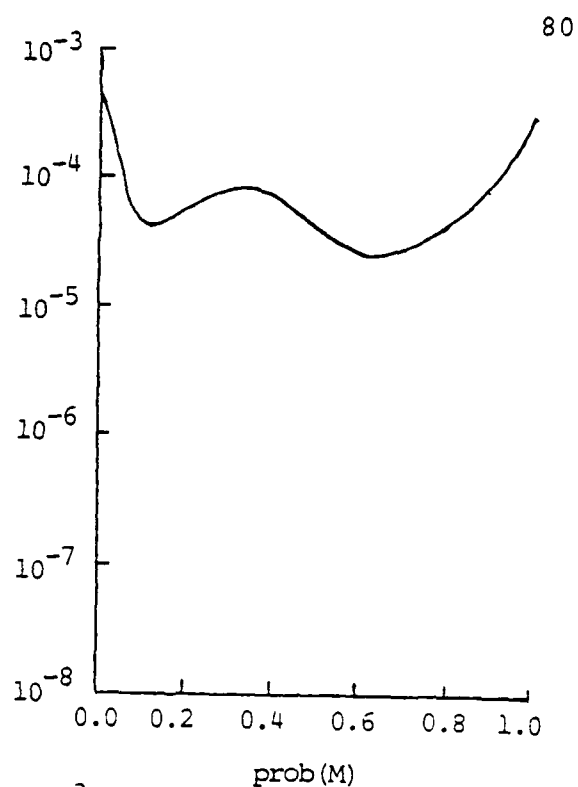
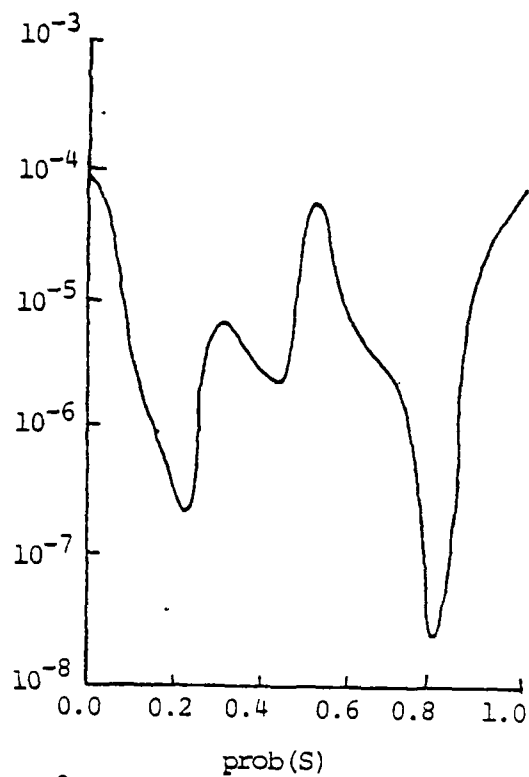


Figure 5.2 P_{esc} , state A

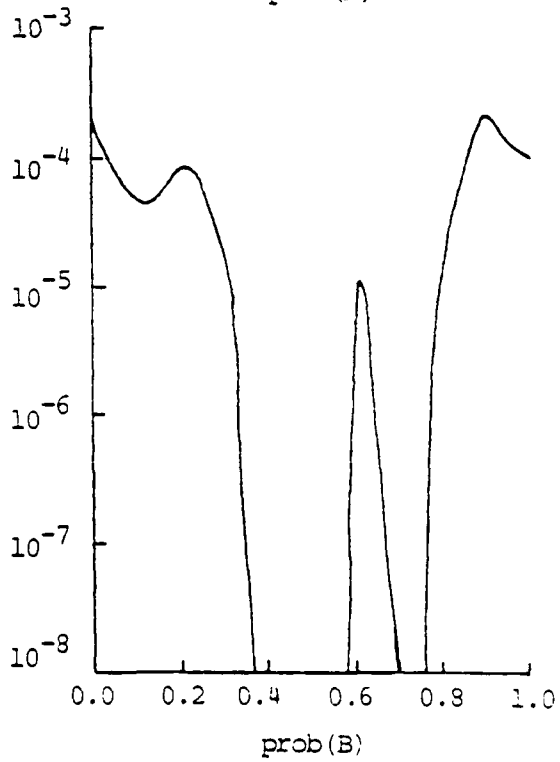
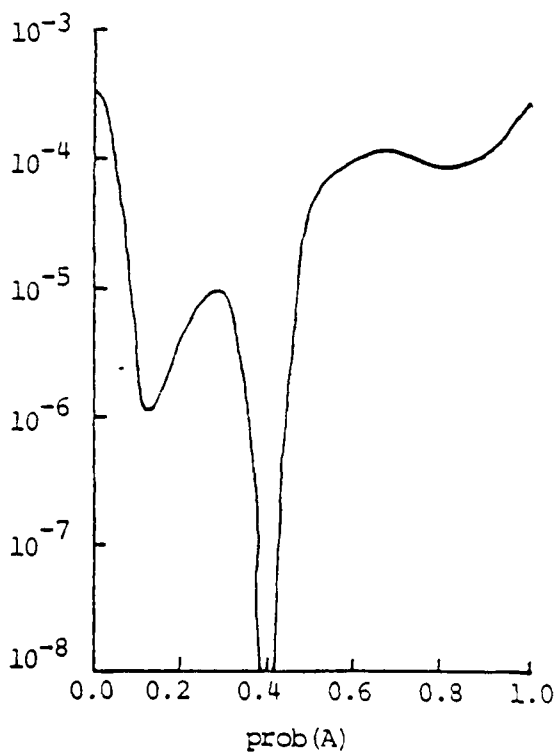
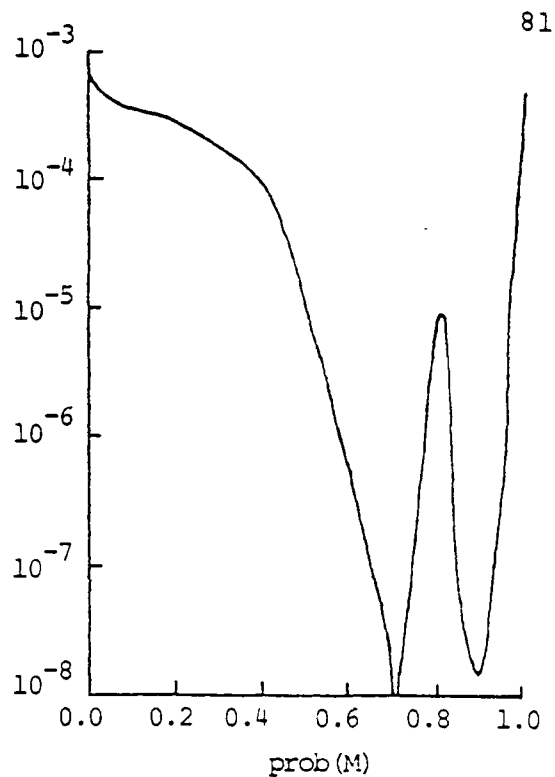
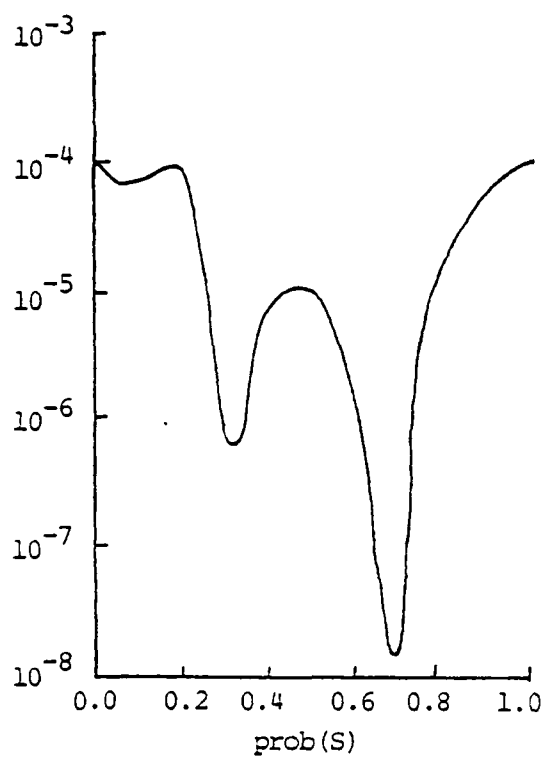
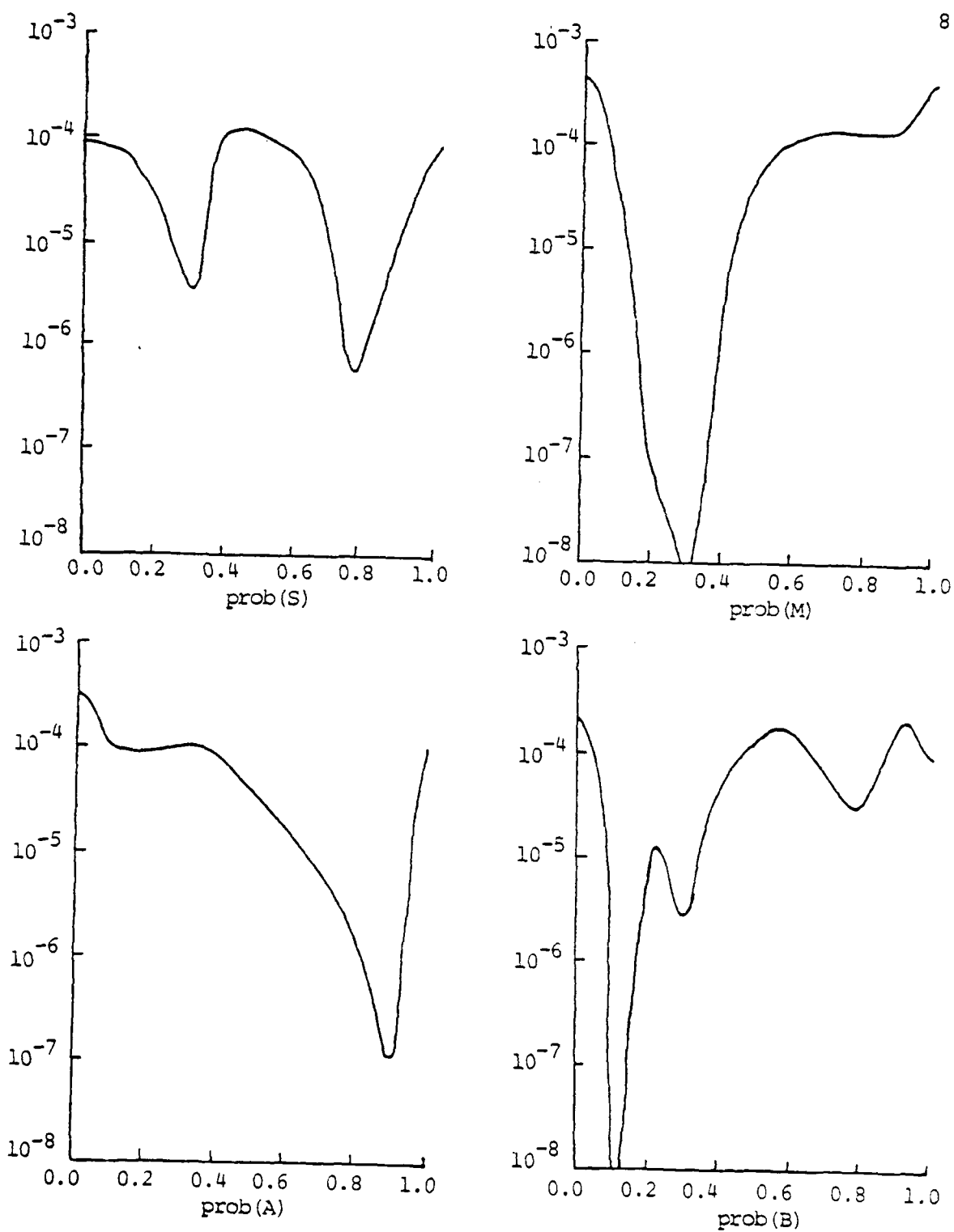
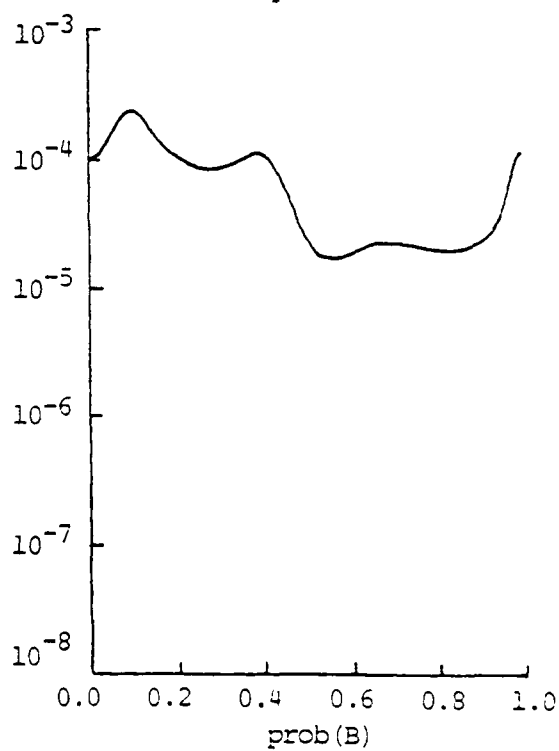
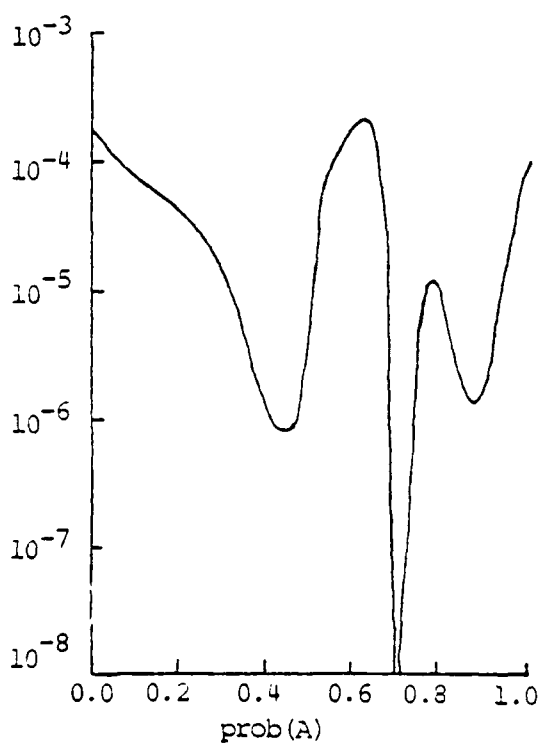
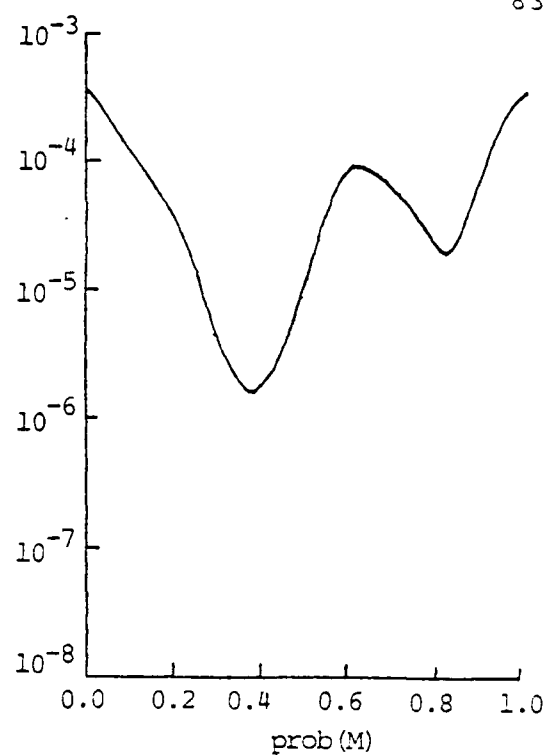
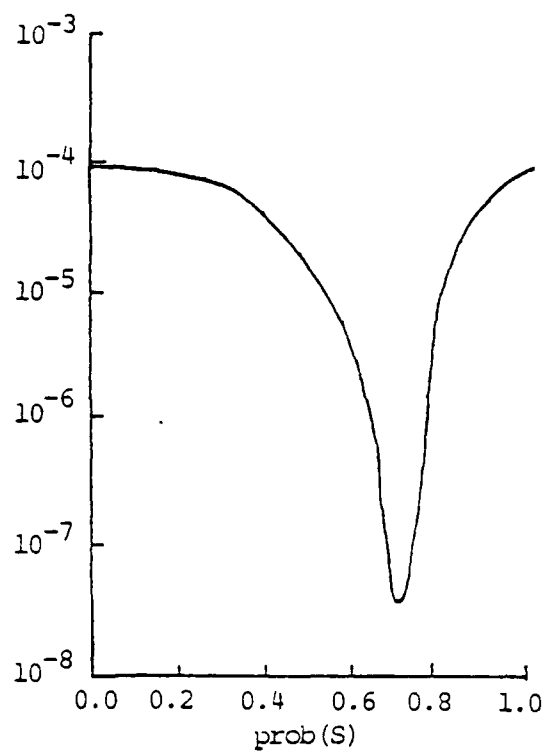


Figure 5.3 P_{esc} , state B

Figure 5.4 P_{esc} , state C

Figure 5.5 P_{esc} , state D

In addition, the results show that in some regions where the probability of escape is high for some states it is low for others. This suggests monitoring the statistics of a number of states (outputs) simultaneously to achieve a low escape probability over a wide region of input statistic variation. For example, by monitoring both states A and B, the escape probability will be low as the statistics of pin B varies in the range $0.1 \leq b \leq 0.9$.

5.3 Test Simulations for the Shift Register

The results obtained for the escape probability of the shift register have been used to specify input statistics for the computer simulation of statistical testing of the device. From Figure 5.6 which shows the escape probability of state D versus the A input statistic we observe that for $a=0.6$ $P_{esc}=1.8 \times 10^{-4}$ and for $a=0.7$ $P_{esc}=2.58 \times 10^{-11}$. Both cases were simulated and the results show that all faulty configurations failed the test for $P_{esc}=2.58 \times 10^{-11}$ ($a=0.7$) and faulty configuration 2022 (serial input stuck-at-zero) passed the test when $P_{esc}=1.8 \times 10^{-4}$ ($a=0.6$). The results of the test for the other states and also their escape probabilities for each case are given in Table 5.3. It is interesting to note that for an increase of 16 percent of the input statistic A the escape probability decreased from 1.8×10^{-4} to 2.58×10^{-11} .

Table 5.3 Escape probability and results of test simulation for two-bit shift register

<u>State</u>	<u>Escape Probability</u>	<u>Faulty Configuration Passing Test</u>
A	6.0×10^{-5}	2122
B	1.1×10^{-4}	2022
C	7×10^{-6}	None
D	2.58×10^{-11}	None

(a) Pin A statistic, $a=0.7$

<u>State</u>	<u>Escape Probability</u>	<u>Faulty Configuration Passing Test</u>
A	5.8×10^{-5}	2122
B	10^{-4}	None
C	2.5×10^{-5}	None
D	1.8×10^{-4}	2022

(b) Pin A statistic, $a=0.6$

5.4 Probability of Escape of a Four-Bit Synchronous Counter

The feasibility of testing a four-bit synchronous counter by statistical methods is investigated in this section by examining the probability of escape of a faulty circuit. Figure 5.6 shows the input and output pins of a typical programmable four-bit synchronous counter.

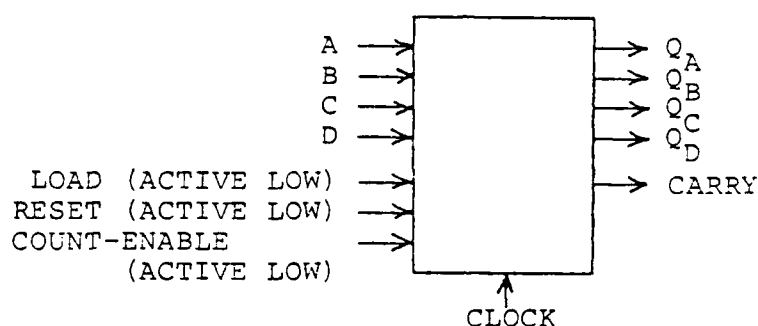


Figure 5.6 Four-bit synchronous counter

The input pin statistics are given in Table 5.4. Using these input statistics, an experiment with a test length of $N=10,000$ and a test stringency of $\epsilon=0.01$ was defined. We then obtained the escape probability of state ϵ (Q_A, Q_B, Q_C, Q_D all zero) using the computer programs given in the appendix. The results were plotted against the

Input Pin	Probability
A	0.5
B	0.3
C	0.3
D	0.1
LOAD (L)	0.4
RESET (R)	0.9
COUNT-ENABLE (D)	0.9

statistics of a pin varying between zero and one while the statistics of the remaining pins were held constant.

These plots appear in Figures 5.7a through 5.7d.

From Figure 5.7b it is noted that extremely low escape probabilities can be achieved over a wide range of enable input statistic variation. This suggests that in order to screen faulty circuits in an offline testing environment the statistics of PT (count enable) should be selected such that $0.4 \leq pt \leq 0.7$ or $pt \leq 0.2$. In an on-line testing environment where no control of input statistics is possible, the computation procedures developed can be a guide to the feasibility and efficiency of statistical on-line testing.

5.5 Computer Simulations

From the plots of Figure 5.7 input statistics are selected so that for one set of statistics the escape probability is $P_{esc}=5.10^{-3}$ and for another set $P_{esc}=1.7 \times 10^{-23}$. These statistics are given in Table 5.5.

Table 5.5 Input statistics

$P_{esc}=1.7 \times 10^{-23}$	$P_{esc}=5 \times 10^{-3}$
R=0.9	R=0.9
L=0.4	L=0.4
P=0.5	P=0.9
A=0.5	A=0.5
B=0.3	B=0.3
C=0.3	C=0.3
D=0.1	D=0.9

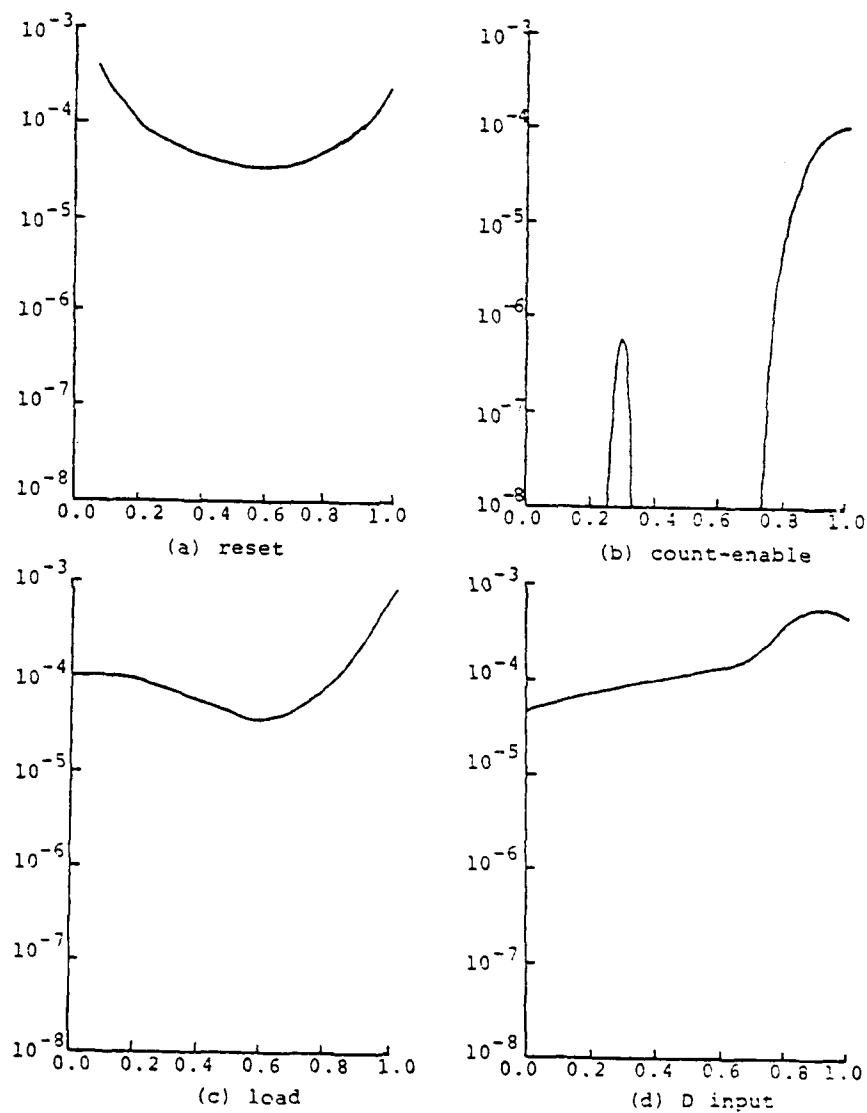


Figure 5.7 Escape probability

Faulty configurations covering all single pin faults and some multiple pin faults which caused minimum deviation from the fault-free input statistics were simulated. The results of the test are tabulated in Table 5.6 for both cases. For the first case where $P_{esc}=1.7 \times 10^{-23}$ none of the faulty circuits passed the test. However, in the second case, forty-three percent of the single pin faulty configurations passed the test ($P_{esc}=5.10^{-3}$). These results are consistent with the calculated escape probabilities for the specified input statistics.

Table 5.6 Simulation results - faulty configurations passing test

$P_{esc}=1.7 \times 10^{-23}$	$P_{esc}=5 \times 10^{-3}$		
None	2222202	B	s-a-0
	2222220	A	s-a-0
	2122222	L	s-a-1
	2212222	P	s-a-1
	2222212	B	s-a-1
	2222221	A	s-a-1

6 CONCLUSIONS AND FURTHER RESEARCH

In this study we have shown that the statistical testing strategy outlined in Chapter 1 for testing sequential circuits provides excellent fault detection under specified assumptions and input statistics. The assumptions were that faults occur at the input pins of the sequential circuits and that the input lines are statistically independent and stationary. The justification for these assumptions was provided.

As a result of the study, algorithms were derived from which the test quality for given test parameters (number of samples and stringency) could be obtained. Using these algorithms, input statistics corresponding to high test qualities were found. These input statistics were then used in computer simulations of the test for specified faulty circuits. It was found that the predicted high test quality for the given input statistics were verified by the computer test simulations.

Furthermore, investigations were done in various methods for obtaining output statistics as a function of input statistics from the behavioral description of sequential circuits. Using the Markov analysis, the output statistics of an n -bit synchronous programmable counter were obtained. Also, methods for obtaining output statistics directly from the network realization were developed. In one method, the output statistics were obtained directly

from the circuit using regular expressions. Another method took advantage of the iterative structure of some networks to obtain output statistics.

Further research: An important assumption made in the study was that the input statistics to sequential circuits under test were independent and stationary. Hence, a statistical analysis of real-time digital signals is needed in order to justify the application of the results of this study to real-time applications. The case of statistical testing and the standardness of its approach including the results of this study should provide good motivation for work in this area.

A further outcome of this study is the possibility of automating the procedure of obtaining output statistics directly from the circuit using regular expressions. Given such a procedure, faults can easily be injected into the circuit and the faulty output statistics derived. This would facilitate the derivation of test quality measurements for circuits in which the pin-fault assumption is not good.

In the study the effect of intermittent faults on the test was not examined. Further work may be done, for example, regarding the effect of the duration of intermittent faults on the output statistics of sequential circuits.

Finally, the algorithms obtained for the derivation of test quality measurements of sequential circuits are directly applicable to combinational circuits. Moreover, the procedures derived in the study can be used to design effective offline tests also.

7 LIST OF REFERENCES

- [1] Alberts, R.D., J.B. Clary, J.W. Gault, S.J. Weikel, and R.A. Whisnant. 1977. A study of a standard bit circuit. System Instrumentation Department, Research Triangle Institute, RTP, NC. February.
- [2] Booth, T.L. 1967. Sequential Machines and Automata Theory. John Wiley and Sons, Inc., New York, NY.
- [3] Brzozowski, J.A. 1964. Derivatives of regular expressions. J. Assoc. Comput. Mach. 11(4), October.
- [4] Brzozowski, J.A. 1964. Regular expressions from sequential circuits. IEEE Trans. Comput., vol. C-13, pp. 1101-1103, December.
- [5] Gault, J.W. and S.H. Ardalan. 1978. Referenceless on-line monitoring. Proc. of Gov't. Microcircuits Applications Conf., November.
- [6] Ketelsen, M.L. 1976. An integrated circuit fault model for digital systems. PhD dissertation, Report R-743, Coordinated Science Lab, Univ. of Illinois, Urbana, IL.
- [7] Kohavi, Z. 1970. Switching and finite automata theory. McGraw-Hill, New York, NY.
- [8] Losq, J. 1976. Referenceless random testing. Proc. 1976 Int'l. Symp. on Fault-Tolerant Computing, pp. 108-113, Pittsburgh, PA.
- [9] Parker, K.P. and E.J. McCluskey. 1975. Probabilistic treatment of general combinational networks. IEEE Trans. Comput., vol. C-14, pp. 573-578, May.
- [10] Parker, K.P. and E.J. McCluskey. 1975. Analysis of logic circuits with faults using signal probabilities. IEEE Trans. Comp., vol. C-24, pp. 668-673, June.
- [11] Parker, K.P. and E.J. McCluskey. 1978. Sequential circuit output probabilities from regular expressions. IEEE Trans. Comp., vol. C-27, pp. 222-231, March.
- [12] Shedlesky, J.J. and E.J. McCluskey. 1975. The error latency of a fault in a combinational circuit. Dig. 1975 Int'l. Symp. on Fault Tolerant Computing, June.

- [13] Shedlesky, J.J. and E.J. McCluskey. 1976. The error latency of a fault in a sequential digital circuit. IEEE Trans. Comp., vol. C-25, pp. 655-659, June.
- [14] Shedlesky, J.J. 1977. Random testing: practicality vs. verified effectiveness. Digest 1977 Int'l. Symposium on Fault Tolerant Computing, pp. 175-179, June.
- [15] Suarez, M.S., P. Deschizeaux, G. Nicoud, and F. Martin. 1976. Statistical fault-location in logical circuits. Proc. FTCS-76, June.
- [16] Tellez-Giron, R. and R. David. 1974. Random fault-detection in logical networks. IFAC Int'l. Symp. on Discrete Systems Dig., Riga, USSR, pp. 232-241, October.

A P P E N D I C E S

APPENDIX A Computation and Simulation Programs

Introduction

This appendix contains all listings of the programs required for the computation of escape probabilities and for the simulation of tests for the two-bit right-shift parallel-load shift register and the four-bit programmable counter. The program for obtaining α , the coefficient of the exponent required for $G(f)$, the pin-fault distribution function, is also provided. Table A.1 contains a glossary of the main programs and the subroutines required by each program. Example input data are provided for each program in the listings.

Table A.1 Main programs and subroutines

<u>Program or Subroutine</u>	<u>Definition</u>
ALPHA	Computes exponent coefficient for pin-fault distribution function given number of input pins.
BICOFF	Routine that generates binary coefficients of a decimal number.
CNSTAT	Routine that computes state probabilities of an N-bit programmable counter.
CNTRA	Routine that simulates an N-bit programmable counter under random inputs.
CNTRSIM	Program that simulates the statistical testing of a programmable counter with specified pin faults.
PESC	Routine that computes probability of escape given the fault density function, test length and test stringency.
PESCAN	Program that generates the fault density function of a given sequential circuit under the statistical pin-fault model and calculates escape probabilities.
RAND	Routine that generates random one with specified probability.
SHIFS	Routine that simulates a 2-bit right-shift parallel-load shift register under random inputs.
SHIFT	Routine that computes state probabilities of a 2-bit right-shift parallel-load shift register.
SHIFTSIM	Program that simulates the statistical testing of a 2-bit right-shift parallel-load shift register with specified pin faults.

```

// EXEC FIGCG
//C.SYSIN DD *
C      PROGRAM NAME PESCAN
C      *****
C *
C * THIS PROGRAM COMPUTES THE FAULT DENSITY FUNCTION OF AN N-BIT PROGR-
C * AMMABLE COUNTER. THE PROBABILITY OF ESCAPE IS ALSO COMPUTED FOR A
C * SPECIFIED TEST LENGTH AND STRINGENCY.
C * THE PROGRAM IS EASILY MODIFIED SUCH THAT ANY SEQUENTIAL CIRCUIT CAN
C * BE ANALYSED. ONLY THE FUNCTION RELATING OUTPUT TO INPUT STATISTICS
C * NEED BE SPECIFIED. IN THE PROGRAM:
C * AIN IS AN ARRAY CONTAINING THE INPUT STATISTICS.
C * NP IS THE NUMBER OF PINS.
C * NBIT IS THE NUMBER OF BITS( SAY, # OF PRESET INPUTS).
C * ALPHA IS THE EXPONENT CCEFF. OF THE PIN FAULT PROBABILITY FUNCTION
C * G(K) FOR NP PINS.
C * CNSTAT IS A ROUTINE THAT COMPUTES THE STATE STATISTICS GIVEN INPUT
C * STATISTICS FOR AN N-BIT PROG. COUNTER.
C * IX IS THE TEST LENGTH AND DELTA IS THE STRINGENCY.
C * FOR A FAULTY CONFIGURAT ON CORRESPONDING TO 0<K<=3**NP-1, G(K) IS
C * THE PROBABILITY OF THAT FAULTY CONFIGURATION OCCURING.
C * ZF(K) IS THE OUTPUT STATISTIC CORRESPONDING TO FAULTY CONFIGURATION
C * K. PHI(Z) IS THE ARRAY CONTAINING THE FAULT DENSITY FUNCTION.
C * PESC IS A ROUTINE THAT COMPUTES THE ESCAPE PROBABILITY GIVEN THE
C * FAULT DENSITY FUNCTION, THE TEST LENGTH, TEST STRINGENCY AND FAULT-
C * FREE OUTPUT STATISTIC.
C *
C * SASAN ARDALAN. SUMMER, 1978.
C *
C      *****
0001      DIMENSION FFREE(256)
0002      DIMENSION PIK(21,101)
0003      DIMENSION INF(8,35,8),LN(8)
0004      DIMENSION BC(16),AIN(256),S(256),PC(256)
0005      DIMENSION AIN2(8),B(8)
0006      DIMENSION PHI(1001),F(1001)
0007      DIMENSION G(3000),ZF(3000)
0008      INTEGER A(8)
0009      REAL L
0010      DATA AIN/0.9,0.4,0.9,0.1,0.3,0.3,0.5/
0011      DATA FFREE/0.2297,0.20039,0.12028,0.0934,0.084315,0.0813,
      $0.05,0.0395,0.02599,0.0226,0.0135,0.0101,0.00934,0.0090,
      $0.0055,0.0043/
C
C      DATA FOR CALCULATION OF PHI(Z)
C
0012      ARG=2.0/0.1040893
0013      ALPHA=ALCG(ARG)
C
C      NPX1 IS THE NUMBER OF INCREMENTS FOR THE INPUT STATISTIC VAF
C      DX IS THE INCREMENT VALUE.
C      NPZ IS THE NUMBER OF ELEMENTS IN PHI(Z)
C
0014      NPZ=1000
0015      NPX1=11
0016      DX=1.0/(NPX1-1)
C
C      NP=NUMBER OF PINS
C

```

```

0017      NP=7
0018      NBIT=4
0019      N=1
0020      DO 2 I=1,NBIT
0021      2 N=N*2
0022      DO 1050 ISTATE=1,1

C
C      VARY INPUT STATISTIC CORRESPONDING TO AIN(JL) WHILE KEEPING
C      REMAINING INPUT STATISTIC FIXED. ALLOWS THE PROB. OF ESCAPE
C      BE OBTAINED AS A FUNCTION OF AN INPUT STATISTIC.
C
0023      DO 1040 JL=1,4
0024      TEMP=AIN(JL)
0025      DO 1020 IG=1,NPX1
0026      AIN(JL)=(IG-1)*DX
0027      F=AIN(1)
0028      L=AIN(2)
0029      PT=AIN(3)
0030      DO 571 MI=1,NBIT
0031      571 AIN2(MI)=AIN(3+MI)

C
C      COMPUTE FAULT-FREE STATISTICS.
C
0032      CALL CNSTAT(R,L,PT,FFREE,NBIT,PC,BC,RES1,AIN2,N)
0033      ZJ=FFREE(ISTATE)
0034      WRITE(3,776)
0035      776 FORMAT(/,10X,'**** INPUT PROBABILITY VECTOR ****'/)
0036      WRITE(3,777)(AIN(NK),NK=1,NP)
0037      777 FORMAT(10X,F5.5/)
0038      DO 5 K=1,NP
0039      5 LN(K)=0
0040      NF=1
0041      DO 6 J=1,NP
0042      6 NF=NF*2
0043      NF=NF-1
0044      DO 10 I=1,NF
0045      CALL BICCFE(1,NP,BC)
0046      NB=0
0047      DO 15 K=1,NP
0048      IF(BC(K).EQ.0)NB=NB+1
0049      15 CONTINUE
0050      LN(NB)=LN(NB)+1
0051      DO 20 K=1,NP
0052      20 INF(NB,LN(NB),K)=BC(K)
0053      10 CONTINUE

C
0054      COMPUTE G(K), ZF(K).

C
C      KK=0
C
C      "I" IS THE NUMBER OF PINS AT FAULT.
C
0056      DO 40 I=1,NP
0057      K1=I

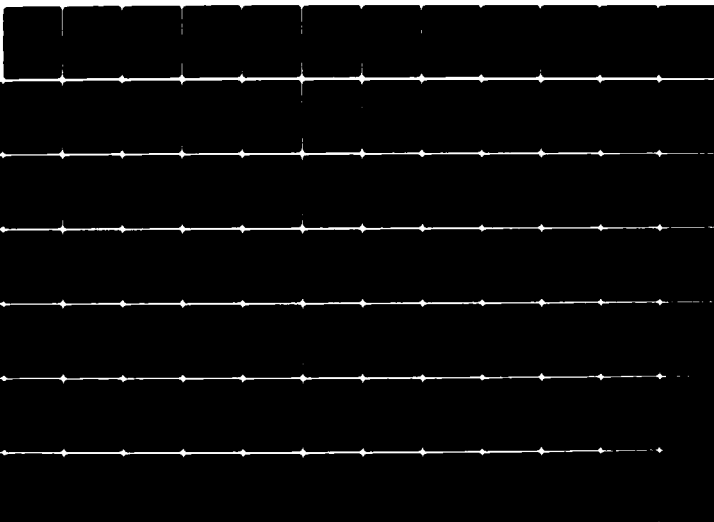
```


NO-8884 888

RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/8 12/1
TECHNIQUES FOR ON-LINE FAULT MONITORING IN MODULAR DIGITAL SYST—ETC(U)
JAN 80 J W SAULT, K S TRIVEDI, P N MARINOS N00039-78-C-0431
RTI/1667/00-01F NL

UNCLASSIFIED

304
802
004 580





```

0058      PRB=EXP(-ALPHA*I)
0059      NK=1
0060      DO 41 K=1,K1
0061      41 NK=NK*2
0062      DO 45 K=1,NK
0063      CALL SICOFF(K,K1,BC)
0064      LNK=LN(I)
0065      DO 46 J=1,LNK
0066      DO 48 IK=1,NP
0067      48 A(IK)=2
0068      IL=0
0069      DO 47 IK=1,NP
0070      IF(INF(I,J,IK).NE.0)GOTO 47
0071      IL=IL+1
0072      A(IK)=SC(IL)
0073      47 CONTINUE
0074      DO 500 MI=1,NP
0075      AR=A(MI)
0076      IF(AIN(MI).NE.0.0)GOTO 499
0077      B(MI)=0.0
0078      IF(AR.EQ.1)B(MI)=1.0
0079      GOTO 500
0080      499 B(MI)=AR/2.0*(1.0+(AR-2.0)*(AIN(MI)-2.0)/AIN(MI))*AIN(MI)
0081      500 CONTINUE
0082      R=B(1)
0083      L=B(2)
0084      PT=B(3)
0085      DO 501 MI=1,NBIT
0086      501 AIN2(MI)=B(3+MI)
0087      C
0088      CALL CNSTAT(R,L,PT,S,NBIT,PC,BC,RES1,AIN2,N)
0089      KK=KK+1
0090      G(KK)=PRB
0091      ZF(KK)=S(ISTATE)
0092      46 CONTINUE
0093      45 CONTINUE
0094      40 CONTINUE
0095      C
0096      C CALCULATION OF PHI(Z)
0097      C
0098      DZ=1.0/NPZ
0099      CZ=CZ/2.0
0100      NPZ1=NPZ+1
0101      WRITE(3,113)ISTATE
0102      113 FORMAT(/10X,'STATE NUMBER:',I6//)
0103      DO 600 I=1,NPZ1
0104      Z=(I-1)*CZ
0105      ZL=Z-DZ
0106      ZH=Z+DZ
0107      SUM=0.0
0108      DO 650 K=1,KK
0109      IF((ZF(K).GT.ZL).AND.(ZF(K).LT.ZH))SUM=SUM+G(K)
0110      650 CONTINUE
0111      PHI(I)=SUM
0112      F(I)=Z
0113      600 CONTINUE
0114      C
0115      C CALCULATE PROBABILITY OF ESCAPE
0116      C
0117      IX=10000
0118      DELTA=0.01
0119      CALL PESCI(PHI,ZJ,DELTA,NPZ1,PRCB,IX)
0120      WRITE(3,116)ZJ,DELTA,IX,PRCB
0121      116 FORMAT(/10X,'ZJ= ',F8.5,' WINDO= ',F7.4,' # OF APPLIED INPU
0122      *VECTORS:',I8//10X,'PROBABILITY OF ESCAPE:',E14.7)
0123      C
0124      C END OF CALCULATION OF PROCB, ESCAPE
0125      C
0126      1020 CONTINUE
0127      AIN(JL)=TEMP
0128      1040 CONTINUE
0129      1050 CONTINUE
0130      STOP
0131      END

```

```

0001      SUBROUTINE CNSTAT(R,L,PT,S,NB,PC,BC,RES1,A,N)
C      *****
C      * THIS ROUTINE COMPUTES THE STATE STATISTICS OF AN N-BIT PROGRAMMABLE **
C      * COUNTER SIMILAR TO THE 74163 GIVEN THE INPUT STATISTICS.      *
C      * R,L,AND PT ARE THE RESET,LOAD,AND COUNT ENABLE INPUT STATISTICS. *
C      * A IS THE ARRAY OF INPUT PRESET STATISTICS LSB FIRST.          *
C      * S IS AN ARRAY CONTAINING THE COMPUTED STATE STATISTICS.        *
C      * NB IS THE NUMBER OF BITS.                                       *
C      * N=2**NB IS AN INPUT.                                           *
C      * PC, BC ARE WORKING ARRAYS.                                       *
C      * RES1 IS THE STATISTIC OF THE STATE ZERO AND IS AN OUTPUT.      *
C      * BICCOFF IS A ROUTINE THAT GENERATES THE BINARY COEFFICIENTS OF A
C      * GIVEN DECIMAL INTEGER.
C      * SASAN ARDALAN. SUMMER. 1978.
C      *****
0002      DIMENSION A(NB),S(N),PC(N),BC(NB)
0003      REAL L,KP
0004      ALFA=1.0-(1.0-PT)*L*R
0005      BETA=R*PT*L
0006      GAMMA=R*(1.0-L)
C      CALCULATION OF PC(K)
C
0007      DO 1 I=1,N
0008      CALL BICCOFF(I,NB,BC)
0009      KP=1.0
0010      DO 2 K=1,NB
0011      2 KP=((1.0-A(K))+(-1.0+2.0*A(K))*BC(K))*KP
0012      PC(I)=KP
0013      1 CONTINUE
C      CALCULATION OF 1.0-GAMMA*SIG0FSIG(BETA/ALFA)K/1.0+SIG(BETA/ALF
C
0014      SM1=0.0
0015      SM3=0.0
0016      BETAL=BETA/ALFA
0017      BDA=1.0
0018      DO 10 I=2,N
0019      BDA=BDA*BETAL
0020      SM1=SM1+BDA
0021      I1=N-I+1
0022      BLA=1.0
0023      DO 25 M=1,I1
0024      25 BLA=BLA*BETAL
0025      SM3=SM3+PC(I1)*(1.0-BLA)
0026      10 CONTINUE
0027      SM3=SM3/(1.0-BETAL)
0028      RES1=(1.0-GAMMA*SM3/ALFA)/(1.0+SM1)
0029      DO 50 J=2,N
0030      BAJ=1.0
0031      DO 23 LL=2,J
0032      23 BAJ=BAJ*BETAL
0033      SM1=0.0
0034      IF(BETAL.EQ.0)GOTO 32
0035      DO 30 K=2,J
0036      K1=J-K+1
0037      BCAK=1.0
0038      DO 31 M=1,K1
0039      31 BCAK=BCAK*BETAL
0040      BCAK=BCAK/BETAL
0041      30 SM1=SM1+BDAK*PC(K1)
0042      32 RES2=SM1*GAMMA/ALFA
0043      S(J)=BAJ*RES1+RES2
0044      50 CONTINUE
0045      S(1)=RES1
0046      RETURN
0047      END

```

```

0001      SUBROUTINE SICOFF(I,NB,BC)
C      *****
C      *
C      * THIS ROUTINE GENERATES THE BINARY COEFFICIENTS OF A DECIMAL INTEGER *
C      * I. I IS THE DECIMAL INTEGER INPUT. *
C      * BC IS AN ARRAY CONTAINING BINARY COEFFICIENTS( MSB FIRST). *
C      *  $I = BC(1) + 2*BC(2) + 4*BC(3) + 8*BC(4) + \dots + 2^{(NB-1)}*BC(NB)$ . *
C      *
C      *****
0002      DIMENSION BC(1)
0003      M=I-1
0004      N=1.0
0005      DO 10 K=1,NB
0006      .10 N=N*2.0
0007      DC 20 K=1,NB
0008      N=N/2.0
0009      IF(M.LT.N)GO TO 1
0010      BC(K)=1.0
0011      M=M-N
0012      GO TO 20
0013      1 BC(K)=0.0
0014      20 CONTINUE
0015      RETURN
0016      END

```

```

0001      SUBROUTINE PESC(PHI,ZJ,DELTA,N,PFOR,IX)
C      *
C      * THE FOLLOWING SUBROUTINE CALCULATES THE PROBABILITY OF ESCAPE.
C      * PHI IS AN ARRAY WITH N ELEMENTS CONTAINING THE FAULT DENSITY
C      * FUNCTION.
C      * ZJ IS THE FAULT-FREE OUTPUT STATISTIC.
C      * IX IS THE TEST LENGTH.
C      * DELTA IS THE TEST STRINGENCY.
C      * PROB IS THE ESCAPE PROBABILITY.
C      * SASAN ARDALAN. SPRING 1978
C      *
C      *
0002      DIMENSION PHI(1)
0003      N1=N+1
0004      ZINC=1.0/N
0005      PROB=0.0
0006      IK=IX*(ZJ-DELTA)
0007      LK=IX*(ZJ+DELTA)
0008      TPX=2.0*3.1415926*IX
0009      IF((IK.LT.0).OR.(LK.GT.1X))RETURN
0010      DO 30 I=IK,LK
0011      RK=I
0012      PSUM=0.0
0013      DO 35 K=2,N
0014      IF(PHI(K).EQ.0.0)GO TO 35
0015      FK=K-I
0016      FN=N
0017      ZZ=FK/FN
0018      ZZM=ZZ*(1.0-ZZ)
0019      TP=TPX*ZZM
0020      TEX=(RK-IX*ZZ)*(RK-[X*ZZ]/2.0/IX/ZZM
0021      IF(TEX.GT.100.0)GO TO 35
0022      TEX=-TEX
0023      TEX=EXP(TEX)/SORT(TP)
0024      PSUM=PSUM+PHI(K)*TEX
0025      35 CONTINUE
0026      PROB=PROB+PSUM*ZINC
0027      30 CONTINUE
0028      RETURN
0029      END

```

```

C      PROGRAM NAME CNTRSIM
C      *****
C      *
C      * THIS PROGRAM SIMULATES AN N-BIT PROGRAMMABLE COUNTER SIMILAR TO THE
C      * 74163. THE INPUTS TO THE PROGRAM ARE AS FOLLOWS:
C      * CARD#1: INPUT STATISTICS NB(# OF BITS),R(RESET),L(LOAD),PT(COUNT EN-
C      * ABLE) FORMAT 12,F8.5,2F10.7
C      * CARD#2: PARALLEL LOAD INPUT STATISTICS CONTAINED IN ARRAY A. LSB
C      * ENTERED FIRST. FORMAT 8F10.7
C      * CARD#3: TEST LENGTH( NUMBER OF SIMULATIONS),NSIM. FORMAT I10
C      * CARD#4: STRINGENCY WINDOW,DELTA. FORMAT F10.7
C      * CARD#5: FAULT CONFIGURATION FOR PINS R,L,PT,LSB,....,MSB.
C      * '2' FAULT FREE, '1' STUCK-AT-ONE, '0' STUCK-AT-ZERO. FORMAT 8011.
C      *
C      *
C      * PROGRAM TERMINATES IF 5 DETECTED IN FIRST COLUMN.
C      * OUTPUT CONSISTS OF RESULT OF TEST( PASS/FAIL), SIMULATED AVERAGES,
C      * FAULT FREE STATISTIC FOR EACH STATE.
C      *
C      * CNSTAT: ROUTINE THAT COMPUTES STATE STATISTICS GIVEN INPUT STATISTICS
C      * CNTRS: ROUTINE THAT SIMULATES COUNTER GIVEN INPUT STATISTICS AND
C      * NUMBER OF SIMULATIONS. OUTPUTS SIMULATED STATE AVERAGES.
C      * BIGOFF: ROUTINE REQUIRED BY CNSTAT.
C      * RAND: ROUTINE REQUIRED BY CNTRS.
C      *
C      * SASAN ARCALAN. SUMMER, 1978.
C      *****
0001      DIMENSION A(256),S(256),IA(8),IAS(8),FFREE(206)
0002      DIMENSION T(16),B(16)
0003      DIMENSION PC(16),BC(8)
0004      INTEGER F(16)
0005      REAL L
0006      DATA IAS/97335,44317,65255,76655,90999,93765,39475,20943/
0007      DATA FFREE/0.22873,0.20039,0.12028,0.0934,0.084315,0.0513,
      $0.05,0.0395,0.02699,0.0228,0.0135,0.0104,0.00934,0.00904,
      $0.0055,0.0043/
0008      READ(1,301)NB,R,L,PT
0009      READ(1,302)(A(K),K=1,NB)
0010      300 FORMAT(110)
0011      301 FORMAT(12,F8.5,2F10.7)
0012      302 FORMAT(8F10.7)
0013      WRITE(3,320)
0014      320 FORMAT(//10X,'NB,R,L,PT: '//)
0015      WRITE(3,301)NB,R,L,PT
0016      WRITE(3,322)
0017      322 FORMAT(//10X,'      A,B,C,....',//)
0018      WRITE(3,302)(A(K),K=1,NB)
0019      N=1
0020      DO 19 JJ=1,NB
0021      19 N=N*2
C
C      COMPUTE FAULT FREE STATISTICS.
C
0022      CALL CNSTAT(R,L,PT,FFREE,NB,PC,BC,RES1,A,N)
0023      T(1)=F

```

```

0024      T(2)=L
0025      T(3)=PT
0026      DC 520 KK=1,NB
0027      T(3+KK)=A(KK)
0028
0029      520 CONTINUE
0029      NP=NB+3
0030      READ(1,300)NSIM
0031      READ(1,350)DELTA
0032      WRITE(3,310)NSIM,DELTA
0033      310 FORMAT(1H1,10X,' NUMBER OF SIMULATIONS:',110,' WINDOW:',F10.
0034      85 CONTINUE
0035      WRITE(3,330)
0036      330 FORMAT(/2(30(' ***'),/)/)
0037      READ(1,360)(F(K),K=1,NP)
0038      IF(F(1).GT.2)STOP
0039      350 FORMAT(F10.7)
0040      360 FORMAT(60I1)
0041      DO 500 M=1,NP
0042      AP=F(M)
0043      IF(T(M).NE.0.0)GOTO 499
0044      R(M)=0.0
0045      IF(F(M).EQ.1)B(M)=1.0
0046      GOTO 500
0047      499 B(M)=AP/2.0*(1.0+(AP-2.0)*(T(M)-2.0)/T(M))*T(M)
0048      500 CONTINUE
0049      DO 501 M=1,NB
0050      501 A(M)=B(3+M)
0051      R=B(1)
0052      L=B(2)
0053      PT=B(3)
0054
0054      C
0055      C      SIMULATE FAULTY CONFIGURATION.
0056      C
0056      CALL CNTRS(F,L,PT,A,S,NB,IA,IAS,NSIM)
0057      NS=1
0057      DO 49 I=1,NB
0058      49 NS=NS*2
0059      C      DO 50 I=1,NS
0059      C      DO 50 I=1,3
0059      C
0059      C      PERFORM TEST ON SIMULATED AVERAGES.
0059      C
0059      ZL=FFREE(1)-DELTA
0060      ZH=FFREE(1)+DELTA
0061      IF((S(I).GE.ZL).AND.(S(I).LE.ZH))GOTO 51
0061      C
0061      PRINT TEST RESULTS FOR EACH STATE.
0061
0061
0061      -
0063      WRITE(3,370)(F(K),K=1,NP)
0064      370 FORMAT(/10X,'FAULTY CONFIGURATION ',7I1,' FAILED THE TEST')
0065      GO TO 50
0066      51 WRITE(3,372)(F(K),K=1,NP)
0067      372 FORMAT(/10X,'FAULTY CONFIGURATION ',7I1,' PASSED THE TEST')
0068      50 WRITE(3,375)I,S(I),FFREE(I)
0069      375 FORMAT(/10X,'STATE #',16,' STATE PROB.',F10.7,' FAULT FREE P
0070      S,F10.7,/,30X,70(' '),/)
0071      GOTO 85
0072      STOP
0072      END

```



```
0001      ..      SUBROUTINE RAND(P,J,IX)
C      .....
C      *
C      * THIS ROUTINE GENERATES A RANDOM ONE WITH PROBABILITY P.
C      * J IS THE RANDOM ONE OUTPUT.
C      *
C      .....
0002      CALL RANDU(IX,IY,YFL)
0003      J=1
0004      IF(YFL.GT.P)J=0
0005      IX=IY
0006      RETURN
0007      END
```

```

0001      SUBROUTINE CNTRS(R,L,PT,A,S,NB,IA,IAS,NSIM)
C      *
C      * THIS ROUTINE SIMULATES AN N-BIT PROGRAMMABLE COUNTER SIMILAR TO THE *
C      * 74163. R,L,PT ARE THE RESET,LOAD, AND COUNT ENABLE INPUT STATISTICS. *
C      * A IS THE ARRAY CONTAINING THE PRESET INPUT STATISTICS LSB FIRST. *
C      * NB IS THE NUMBER OF BITS( PRESET INPUTS). *
C      * S IS THE ARRAY OF SIMULATED STATE AVERAGES. *
C      * IAS IS AN ARRAY CONTAINING NB RANDOM INTEGERS(3-5 DIGITS). *
C      * IA IS A WORKING ARRAY. *
C      * NSIM IS THE NUMBER OF SIMULATIONS. *
C      * RAND IS A ROUTINE THAT GENERATES A RANDOM ONE WITH SPECIFIED PROB- *
C      * ABILITY. SASAN ARDALAN, SUMMER, 1978. *
C      *
C      *****
0002      DIMENSION A(1),S(1),IA(1),IAS(1)
0003      REAL L
0004      DATA IXR,IXL,IXPT/38329.80973,61131/
0005      N=1
0006      DO 5 I=1,NB
0007      S=N*N*2
0008      DO 15 J=1,N
0009      S(J)=0.0
0010      J=1
0011      DO 55 KK=1,NSIM
C
C      GENERATE RANDOM INPUTS
C
0012      CALL RAND(R,IXR,IXR)
0013      CALL RAND(L,IXL,IXL)
0014      CALL RAND(PT,IXPT,IXPT)
0015      DO 12 K=1,NB
0016      SA=A(K)
0017      IXA=IAS(K)
0018      CALL RAND(SA,IAG,IXA)
0019      IA(K)=IAG
0020      IAS(K)=IXA
0021      12 CONTINUE
0022      IF(IXR.EQ.0)GOTO 1
0023      IF(IXL.EQ.0)GOTO 2
0024      IF(IXPT.EQ.0)GOTO 50
0025      J=J+1
0026      IF(J.EQ.N+1)J=1
0027      GOTO 50
0028      1 J=1
0029      GOTO 50
0030      2 ISUM=0
0031      KE=1.0
0032      DO 20 K=1,NB
0033      ISUM=KE*IA(NB-K+1)+ISUM
0034      KE=KE*2
0035      20 CONTINUE
0036      J=(ISUM+1)
0037      S(J)=S(J)+1.0
0038      55 CONTINUE
0039      DO 25 J=1,N

```

```

0040      25 S(J)=S(J)/NSIM
C        WRITE(3,325)
C        WRITE(3,1001)(S(J),J=1,N)
0041      325 FORMAT(1H1,/,10X,' SIMULATED RESULTS'//)
0042      100 FORMAT(/,10X,E14.7)
0043      RETURN
0044      END

```

EXAMPLE DATA FOR CNTRSIM

```

//G.SYSIN CD *
4 0.9      0.4      0.9
0.9      0.3      0.3      0.5
      10000
0.01
2222222
1222222
2022222
2220222
2202222
2222202
2222220
2122222
2212222
2221222
2222122
2222212
2222221
1212222
1012222
2C20002
2220001
022222
222022
555555
      1000
4 .9      0.4      0.9
0.1      0.3      0.3      0.5
0.01
      1000000
4 .9      0.4      0.9
0.1      0.3      0.3      0.5
/*

```

```

C PROGRAM NAME SHIFTSIM
C *****
C *
C * THIS PROGRAM SIMULATES THE STATISTICAL TESTING OF A TWO BIT PARALLEL
C * LOAD, SHIFT RIGHT, SHIFT REGISTER. INPUTS TO THE PROGRAM ARE:
C * CARD#1: INPUT STATISTICS S,M,A,B FORMAT 8F10.7
C * CARD#2: NUMBER OF SIMULATIONS( TEST LENGTH) FORMAT I10
C * CARD#3: STRINGENCY WINDOW FORMAT F10.7
C * CARD#4: FAULT CONFIGURATION FOR PINS S,M,A,B. THE CONFIGURATION IS
C * SPECIFIED BY '2' FAULT FREE, '1' STUCK-AT-ONE, '0' STUCK-AT-ZERO.
C * THE FORMAT IS 8011. EACH FAULT CONFIGURATION IS SPECIFIED ON A
C * SEPARATE CARD.
C * THE PROGRAM TERMINATES WHEN A 5 IS PUNCHED IN COLUMN 1.
C * PRINTOUT IS CONSISTED OF WHETHER THE CIRCUIT PASSES OR FAILS THE
C * TEST, THE FAULT CONFIGURATION, THE FAULT FREE STATISTIC, AND THE
C * SIMULATED AVERAGE.
C *
C * SHIFTS: SUBROUTINE THAT SIMULATES SHIFT REGISTER
C * SHIFT: SUBROUTINE THAT COMPUTES STATE PROBABILITIES OF SHIFT REG.
C * OTHER ROUTINE REQUIRED RAND.
C * NP=NUMBER OF PINS
C * NB= NUMBER OF BITS
C * DELTA= TEST STRINGENCY WINDOW.
C * F(K): ARRAY CONTAINING FAULT CONFIGURATION.
C * T(K): ARRAY CONTAINING INPUT STATISTICS.
C *
C *****
0001 DIMENSION PP(4),PT(10000)
0002 DIMENSION T(4),S(4),FFREE(4)
0003 DIMENSION B(4)
0004 INTEGER SS(4,2),F(4)
0005 NP=4
0006 NB=2
0007 READ(1,302)(T(I),I=1,4)
0008 302 FORMAT(8F10.7)
0009 READ(1,300)NSIM
0010 300 FORMAT(I10)
0011 READ(1,350)DELTA
0012 WRITE(3,310)NSIM,DELTA
0013 310 FORMAT(1H1,10X,' NUMBER OF SIMULATIONS:',110,' WINDOW:',F10.7)
0014 WRITE(3,333)(T(I),I=1,4)
0015 333 FORMAT(//10X,' INPUT PROBABILITY VECTOR S,M,A,B, '//10X, '(F10.7)
C
C SIMULATE FAULT FREE CIRCUIT.
C
0016 CALL SHIFTS(T,S,NSIM,SS,PP,PT)
0017 CALL SHIFT(PP,FFREE)
0018 85 CONTINUE
0019 WRITE(3,330)
0020 330 FORMAT(//2(30(' **')//))
0021 READ(1,360)(F(K),K=1,NP)
0022 IF(F(1).GT.2)STOP
0023 350 FORMAT(F10.7)
0024 360 FORMAT(8011)
0025 DO 300 MI=1,NP
0026 AF=F(MI)

```

```

0027      IF(T(MI).NE.0.0)GOTO 499
0028      B(MI)=0.0
0029      IF(F(MI).EQ.1)B(MI)=1.0
0030      GOTO 500
0031      499 B(MI)=AR/2.0*(1.0+(AR-2.0)*(T(MI)-2.0)/T(MI))*T(MI)
0032      500 CONTINUE
C
0033      SIMULATE FAULTY CONFIGURATION.

C
0034      CALL SHIFTS(B,S,NSIM,SS,PP,PT)
0035      NS=1
0036      DO 49 I=1,NB
0037      49 NS=NS*2
0038      DO 50 I=1,NS
C
C      CHECK IF SIMULATED OUTPUT WITHIN TEST WINDOW OF FAULT FREE DI
C
0039      ZL=FFREE(I)-DELTA
0040      ZH=FFREE(I)+DELTA
0041      IF((S(I).GE.ZL).AND.(S(I).LE.ZH))GOTO S1
C
C      PRINT TEST RESULTS FOR EACH STATE
C
0042      WRITE(3,370)(F(K),K=1,NP)
0043      370 FORMAT(/10X,"FAULTY CONFIGURATION ",411," FAILED THE TEST",
0044      GO TO 50
0045      S1 WRITE(3,372)(F(K),K=1,NP)
0046      372 FORMAT(/10X,"FAULTY CONFIGURATION ",411," PASSED THE TEST",
0047      50 WRITE(3,375)1,S(I),FFREE(I)
0048      375 FORMAT(/10X,"STATE #",16," STATE PROB.",F10.7," FAULT FREE P
S,F10.7,/,/30X,70(,"")/)
0049      WRITE(3,368)PP
0050      368 FORMAT(/10X,"INPUT MEASURES(S,M,A,B):",4(F7.4,3X))
0051      GOTO 85
0052      STOP
C
0053      END

```

```

0001      SUBROUTINE SHIFTS(PV,S,NSIM,SS)
C      *
C      * THIS ROUTINE SIMULATES A 2-BIT PARALLEL-LOAD, RIGHT-SHIFT,SHIFT
C      * REGISTER. PV IS THE ARRAY CONTAINING INPUT STATISTICS S,M,A, AND B.*
C      * S IS AN ARRAY CONTAINING THE RESULTANT SIMULATED AVERAGES FOR EACH
C      * STATE. NSIM IS THE NUMBER OF SIMULATIONS.
C      * SS IS A TWO DIMENSIONAL WORKING ARRAY.
C      * SUBROUTINE RAND GENERATES A RANDOM ONE WITH SPECIFIED PROBABILITY.
C      *
C      * SASAN ARDALAN SUMMER, 1978.
C      *
C      *
0002      DIMENSION PV(4),S(4)
0003      INTEGER SS(4,2),PS,SR,A,B,M
0004      DATA IM,IS,IA,IB/38329.86973,61131.76229/
0005      SS(1,1)=1
0006      SS(2,1)=4
0007      SS(3,1)=4
0008      SS(2,2)=3
0009      SS(1,2)=2
0010      SS(4,1)=1
C
0011      CALL RAND(RM,M,IM)
0012      CALL RAND(RS,SR,IS)
0013      CALL RAND(RA,A,IA)
0014      CALL RAND(RB,B,IB)
0015      IF(M.EQ.0)GOTO 10
0016      PS=A+B*2+1
0017      IF(PS.EQ.3)PS=4
0018      IF((A*B).EQ.1)PS=3
0019      S(PS)=S(PS)+1.0
0020      GOTO SS
0021      10 PS=SS(PS,SR+1)
0022      S(PS)=S(PS)+1.0
0023      55 CONTINUE
0024      DO 14 I=1,4
0025      14 S(I)=S(I)/NSIM
0026      RETURN
0027      END

```

```

0001      SUBROUTINE SHIFT(T,Q)
0002      .....
0003      *
0004      * THIS SUBROUTINE COMPUTES THE STATE PROBABILITIES OF A 2-BIT PARALLEL-
0005      * LOAD, RIGHT-SHIFT, SHIFT REGISTER.
0006      * Q IS THE ARRAY OF COMPUTED STATE STATISTICS.
0007      *
0008      .....
0009      DIMENSION T(NP),Q(4)
0010      REAL M,MC
0011      S=T(1)
0012      M=T(2)
0013      A=T(3)
0014      B=T(4)
0015      AC=1.0-A
0016      BC=1.0-B
0017      SC=1.0-S
0018      MC=1.0-M
0019      Q(1)=SC*MC-S*SC*MC*MC-SC*A*M*MC+BC*AC*M
0020      Q(2)=S*MC-S*S*MC*MC-S*A*M*MC+BC*A*M
0021      Q(3)=S*S*MC*MC+A*S*MC*M+M*A*B
0022      Q(4)=MC*MC*S*SC+MC*M*SC*A+B*AC*M
0023      RETURN
0024      END

```

EXAMPLE DATA: SHIFTSIM

```

//G.SYSIN DD *
0.6      0.9      0.7      0.7
10000
C.C1
2222
2122
1222
2212
2221
0222
2022
2202
2220
2102
2101
0201
2111
2112
5555
/*

```

```

C *****
C *
C * THIS PROGRAM COMPUTES THE EXPONENT COEFFICIENT, ALPHA, OF THE PIN-
C * FAULT PROBABILITY G(K).
C * NP IS THE NUMBER OF INPUT PINS OF THE PACKAGE(CIRCUIT).
C *
C *****
0001      EXTERNAL FCT,COMB
0002      COMMON NP
0003      IEND=500
0004      EPS=1.0E-4
0005      XLI=0.0
0006      XRI=1.0
0007      DO 20 NP=1,40
C
C      RTM1 CAN BE FOUND IN THE IBM SCIENTIFIC SUBROUTINE PACKAGE.
C
0008      CALL RTM1(X,F,FCT,XLI,XRI,EPS,IEND,IER)
0009      WRITE(3,100)X,F,XLI,XRI,EPS,IEND,IER
0010 100  FORMAT(' ',10X,'X,F,XLI,XRI,EPS,IEND,IER'//5(5X,E14.7)//10X,2
0011      XX=2.0/X
0012      ALPHA=ALOG(XX)
0013      WRITE(3,250)ALPHA,NP
0014 250  FORMAT(/10X,'***** ALPHA *****E14.7,' *** # OF PINS ***
      $13)
0015      20 CONTINUE
0016      STOP
0017      END

0001      FUNCTION FCT(X)
0002      EXTERNAL COMB
0003      COMMON NP
0004      SUM=0.0
0005      XK=1.0
0006      DO 40 I=1,NP
0007      XK=XK*X
0008      SUM=SUM+COMB(NP,I)*XK
0009 40  CONTINUE
0010      FCT=SUM-1.0
0011      RETURN
0012      END

0001      FUNCTION COMB(N,K)
0002      IF(K.EQ.1)COMB=N
0003      IF(K.EQ.N)COMB=1.0
0004      IF((K.EQ.1).OR.(K.EQ.N))RETURN
0005      X=1.0
0006      NMK=N-K
0007      DO 20 I=1,N
0008      X=X*I
0009      IF(I.EQ.K)XK=X
0010      IF(I.EQ.NMK)XNMK=X
0011 20  CONTINUE
0012      COMB=X/XK/XNMK
0013      RETURN
0014      END

```


APPENDIX B Three-Dimensional Plot Routine

Introduction

This appendix contains the program for plotting the three-dimensional variation of state probabilities of a 4-bit programmable counter. Two input statistics are varied while the remaining input statistics are held fixed. Also included is a brief introduction about using SYMVU, the three-dimensional plotting routine.

Plotting Procedure

The three-dimensional plotting is done by a program called SYMVU. A complete description of the capabilities of the program can be obtained from the SYMVU MANUAL available from the Laboratory for Computer Graphics and Spatial Analysis, Howard University, 114 Memorial Hall, Cambridge, Massachusetts, 02138. Below we give a brief description on how a plot can be obtained.

The information to be plotted must be stored on disk where the program SYMVU, based on data specifying size, viewing angle, and distance from object (3-D plot) generates a plot based on this data. Thus, two steps are involved. First a two-dimensional array $PlK(I,J)$ must be stored on disk. The value of $PlK(I,J)$ is the z coordinate while the integers I and J specify the x and y coordinates (see Figure B.1). That is, $x=(I-1)*\Delta x+x_0$, $y=(J-1)*\Delta y+y_0$, where Δx and Δy are the x-axis and y-axis increments and

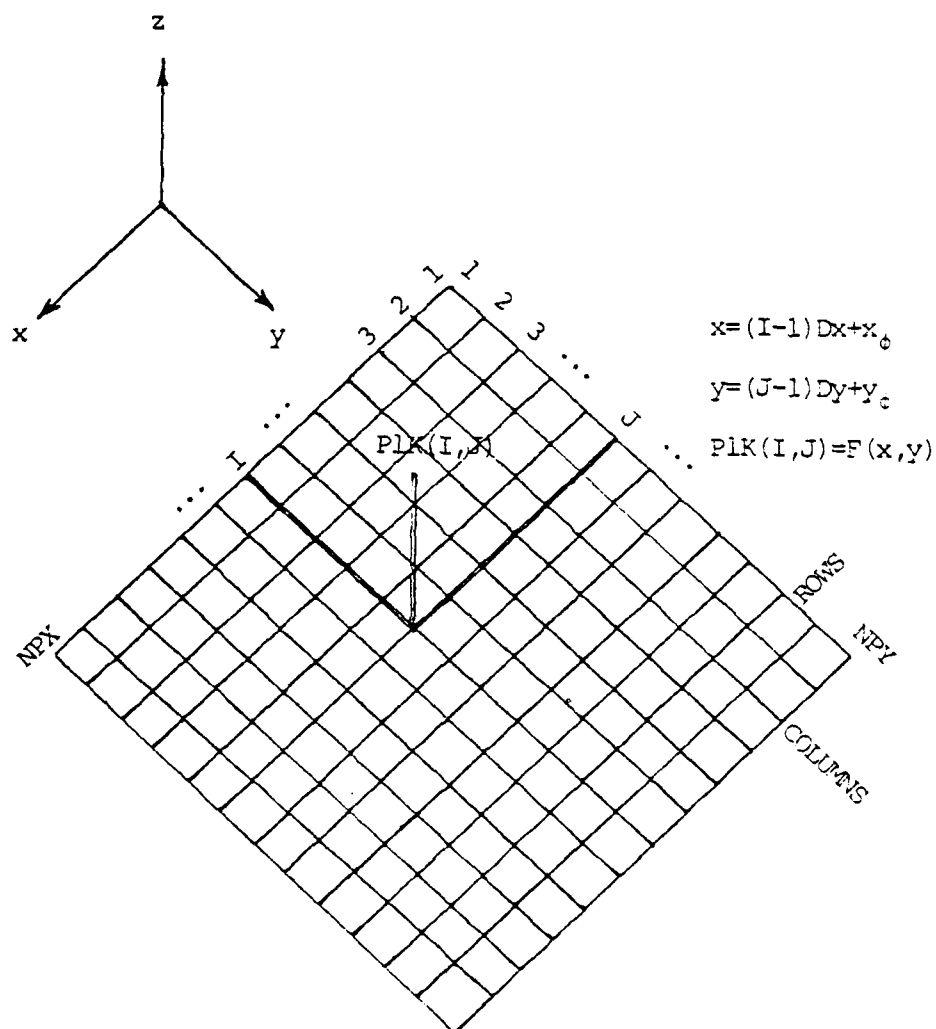


Figure B.1 Three-dimensional plot layout

x_0 and y_0 the starting values. Thus, if the function $z=f(x,y)$ is to be plotted we must calculate the array $PlK(I,J)$ as follows:

```

      DO 500 I=1, NPX
      X = (I-1)*DX + XØ
      DO 500 J=1, NPY
      Y = (J-1)*DY + YØ
500  PlK(I,J)=F(X,Y).

```

Now to store $PlK(I,J)$ on disk we do the following:

```

      WRITE(8) NPX, NPY, RPI, CPI, RMIN, RMAX
      DO 44 I=1, NPY
44  WRITE(8) (PlK(I,J), J=1, NPX)
      REWIND 8

```

where RPI is the number of rows per inch, CPI is the number of columns per inch, RMIN is the minimum value of $z=PlK(I,J)$, and RMAX is the maximum value. Rows correspond to variation along the x axis and columns correspond to variation along the y axis.

Besides the above information transferred to SYMVU by disk, data specifying the title, number of columns and rows, viewing angle, etc. must be provided in the job stream. A typical set of cards required for obtaining a plot are described below.

Card #1 : Contains title of plot up to 72 characters.

Card #2 : Specifies number of rows and columns.

Cols 1-4 : number of rows right justified

Cols 5-8 : number of columns right justified

Col 16 : line type (1 for columns plotted.
2 rows plotted, 4 diagonals
plotted).

Card #3 : Controls viewing angle of plot.

Cols 1-5 : Altitude in degrees (e.g., 45°)

Cols 6-10: Azimuth, the desired horizontal
angle of viewing.

Cols 11- Width in inches along side of plot
15: that is width of x axis.

Cols 3-4 : Height in inches

Cols 26-
35: RMIN minimum z

Cols 36-
45: RMAX maximum z.

Note that columns 68, 72, and 76 must contain a 1 in
Card #2.

The necessary JCL along with example data illustrating
the usage of the above cards is given along with the program
listing. Further information on the extensive capabilities
of SYMVU can be found in the SYMVU manual.

```

// EXEC FTGCG
//C.SYSIN CO *
C   PROGRAM NAME PLOTENT
C   *****
C   *
C   * THIS PROGRAM PLOTS THE STATE PROBABILITY OF A COUNTER VERSUS THE
C   * VARIATION OF TWO OF ITS INPUT STATISTICS WHILE THE OTHER STATISTICS *
C   * REMAIN FIXED.
C   * CNSTAT IS A ROUTINE THAT RELATES STATE STATISTICS TO INPUT STATISTICS
C   * R,L,PT ARE THE RESET, LOAD, AND COUNT ENABLE STATISTICS OF THE COUNTER
C   * A IS AN ARRAY CONTAINING THE PRESET INPUT STATISTICS.
C   * PIK(J,I) IS AN ARRAY CONTAINING THE Z COORDINATE CORRESPONDING TO
C   * X(J), AND Y(I).
C   * NPX IS THE NUMBER OF POINTS ON THE X AXIS.
C   * NPY IS THE NUMBER OF POINTS ON THE Y AXIS.
C   * RMIN AND RMAX ARE THE MIN. AND MAX. VALUES OF PIK(J,I) THAT IS, Z.
C   * ROWS CORRESPOND TO THE X AXIS AND COLUMNS TO THE Y AXIS.
C   * CPI IS THE NUMBER OF COLUMNS/INCH AND RPI THE NUMBER OF ROWS/INCH.
C   * UNIT 8 IS THE DISC WHERE PIK(J,I) IS STORED FROM WHICH THE PROGRAM
C   * SYMVU OBTAINS THE PLOT INFORMATION.
C   *
C   *****
C   DIMENSION PIK(50,50),A(256),PC(256),EC(16),S(256)
C   REAL L
C   READ(1,110)NB
C   READ(1,120)(A(K),K=1,NB)
C   READ(1,130)R,L,PT
110  FORMAT(12)
120  FORMAT(16F5.3)
130  FORMAT(3F5.3)
C   N=1
C   DO 5 I=1,NB
C   N=N+2
C   READ(1,240)NPX,NPY
240  FORMAT(2I3)
C   DX=1.0/NPX
C   DY=1.0/NPY
C   NPX1=NPX+1
C   NPY1=NPY+1
C   DO 500 I=1,NPX1
C   FT=(I-1)*DX
C   DO 500 J=1,NPY1
C   L=(J-1)*DY
C   CALL CNSTAT(R,L,PT,S,NB,PC,EC,RES1,A,N)
C   PIK(I,J)=S(2)
500  CONTINUE
C   NC=NPY1
C   NR=NPX1
C   RPI=2.0
C   CPI=2.0
C   RMIN=0.0
C   RMAX=0.6
C   WRITE(8)NR,NC,RPI,CPI,RMIN,RMAX
C   DO 44 J=1,NR
44  WRITE(8)(PIK(J,I),I=1,NC)
C   FE=10.8
C   CALL EXIT
C   END

```

```

//C.FT08F001 DD DSN=CCRT1.DISP=(NEW,PASS),
//          UNIT=DISK,VOL=SER=RT1222,
// SPACE=(TRK,(S,S),RLSE),
// CCE=(RECFM=VBS,LRECL=2060,BLKSIZE=13000,BUFNO=1)
//G.SYSIN DD *
      4
      C.C  C.3 0.3 0.5
      0.9  C.4 0.9
      40 40
/*
//S2 EXEC SYMVU
//G.NCSPLCT DD SYSCUT=C
//G.SYSIN DD *
      STATS OF CNTR
      40 40 F      2
      45.0340.0 4.0 3.C
      0.6
/*
//G.DATA DD DSN=CCRT1.DISP=OLD
/*

```

1 1 1

SECTION II

PROGRAMMABLE CELLULAR ARRAYS WITH
HARDWARE ENCODED BUILT-IN-TEST (BIT) FACILITIES

by

Dr. Peter N. Marinos
Professor of Electrical Engineering
and Computer Science
Duke University, Durham, N.C.

January 1979

1.0 INTRODUCTION

The advantages of cellular arrays, which result from the regularity of their iterative structure, have been sufficiently documented in previous studies [1-7], and memory manufacturers have fully exploited many of their unique features to produce memories with highly improved device densities, production yields, size, cost, speed, reliability, and noise characteristics. Presently, IC manufacturers are becoming increasingly interested in cellular structures as a means of bringing about further improvements in system integration, reliability, testability, and system maintainability. This, of course, implies incorporation of cellular logic design notions into the design of what has been traditionally known as random logic subsystems which include control units and arithmetic and logic units of digital systems.

There are three main reasons why random logic subsystems have remained largely non-cellular in form, namely: lack of design standardization; inadequate testing procedures for fault detection and fault diagnosis; and poor maintenance schemes in terms of self-reconfiguration in the event of failure, as well as in terms of preventive maintenance. These are major problem areas and are currently attracting a great deal of attention in several industrial and university laboratories. This interest is easily justified by the fact that cost, device densities and yields of mass-produced cellular structures using mature technologies have greatly improved over the years, and we find ourselves now in the comfortable position of being able to build very economically into such cellular structures redundant functional capability which one could use to improve system integration, system testability and maintainability and, in general, system reliability and availability.

In recent publications, Manning [6] describes certain procedures for automatic testing, configuration, and repair of cellular arrays; and Page and Marinos [7] propose a programmable array for use in designing synchronous sequential machines and demonstrate ways for actually embedding arbitrary finite-state machines in such arrays. What is proposed next is a natural extension of these two independent research efforts, and it is based on the strong belief that industry will recognize the many advantages of using cellular structures throughout a digital system once the issues of array standardization

(both functional as well as structural), testability and maintainability have been resolved in a practical and cost-effective manner. This paper describes a programmable cellular array suitable for implementing random logic with built-in-test (BIT) facilities in a way which is a natural extension of the overall system design process. The BIT facility imparts to the cellular structure on-line fault detection and correction capability and is, in effect, a "hardware encoded" version of well-known information encoding procedures relying on the same basic cell used to configure the rest of the cellular structure.

An algorithmic procedure for mapping arbitrary synchronous sequential machines with "hardware encoded" random logic in the cellular array is also presented.

2.0 SEQUENTIAL MACHINE CHARACTERIZATION

A finite state machine is described by the algebraic structure

$$M = \langle X, Z, Q, \delta, \omega \rangle$$

where

X = a finite set of input symbols (x_1, x_2, \dots, x_n) such that

$$x_i \in (0,1), i = 1, 2, \dots, n;$$

Z = a finite set of output symbols (z_1, z_2, \dots, z_p) such that

$$z_j \in (0,1), j = 1, 2, \dots, p;$$

Q = a finite set of states (q_1, q_2, \dots, q_t) defined by the state variables (y_1, y_2, \dots, y_m) such that $m \geq \log_2 t$;

$\delta: X \times Q \rightarrow Q$ is the next-state function

$\omega: \left. \begin{array}{l} X \times Q \rightarrow Z \\ \text{or} \quad Q \rightarrow Z \end{array} \right\}$ is the output function

The general form of the excitation and output functions of a sequential machine may be written as follows:

$$F_r(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m) = \bigcup_{i=1}^t f_{i,r}(x_1, x_2, \dots, x_n) q_i \quad (1)$$

where

$$r = 1, 2, \dots, m, \dots, m + p$$

$$q_i = y_1^* y_2^* \dots y_m^*$$

with y_j^* denoting the state variable y_j either in its complemented or uncomplemented form (i.e., q_i denotes a min-term of the state variables $y_j, j = 1, 2, \dots, m$.)

For reasons outlined elsewhere [7,8], state assignments based on monotone, k-out-of-m codes will be utilized. Among the many advantages offered by these codes, the one of interest in this case is their utility in error detection, and in designing fail-safe sequential machines [9,10]. In view of such a state assignment, one may rewrite equation (1) in the form

$$F_r = \bigcup_{i=1}^t f_{i,r}(x_1, x_2, \dots, x_n) \cdot G_i(m, k) \quad (2)$$

where

$$G_i(m, k) = y_{i1}y_{i2} \dots y_{ik}$$

and $y_{ij} \in (y_1, y_2, \dots, y_m)$, $j = 1, 2, \dots, k$.

Equation (2) suggests a linear array, each cell of which is algebraically described by the function

$$A_{i,r} = g_{i,r} + A_{i-1,r} + f_{i,r}(x_1, x_2, \dots, x_n) G_i(m, k) \quad (3)$$

Figure 2.1 shows the structure of such a cell with an n-input programmable universal logic module (PULM-n) implementing the function $f_{i,r}(x_1, x_2, \dots, x_n)$. The PULM-n unit is programmed via the associated programming register. The linear cellular array shown in Figure 2.2 implements the function F_r given by expression (2), and the two-dimensional cellular structure given in Figure 2.3 illustrates a cellular array capable of implementing any required, finite number of excitation and output fractions F_r necessary for the implementation of a finite state, synchronous sequential machine. One of the structural advantages of the array shown in Figure 2.3 is its ability to support finite state, synchronous sequential machines of arbitrary "state-space" cardinality without requiring any structural changes in the basic cell of the array.

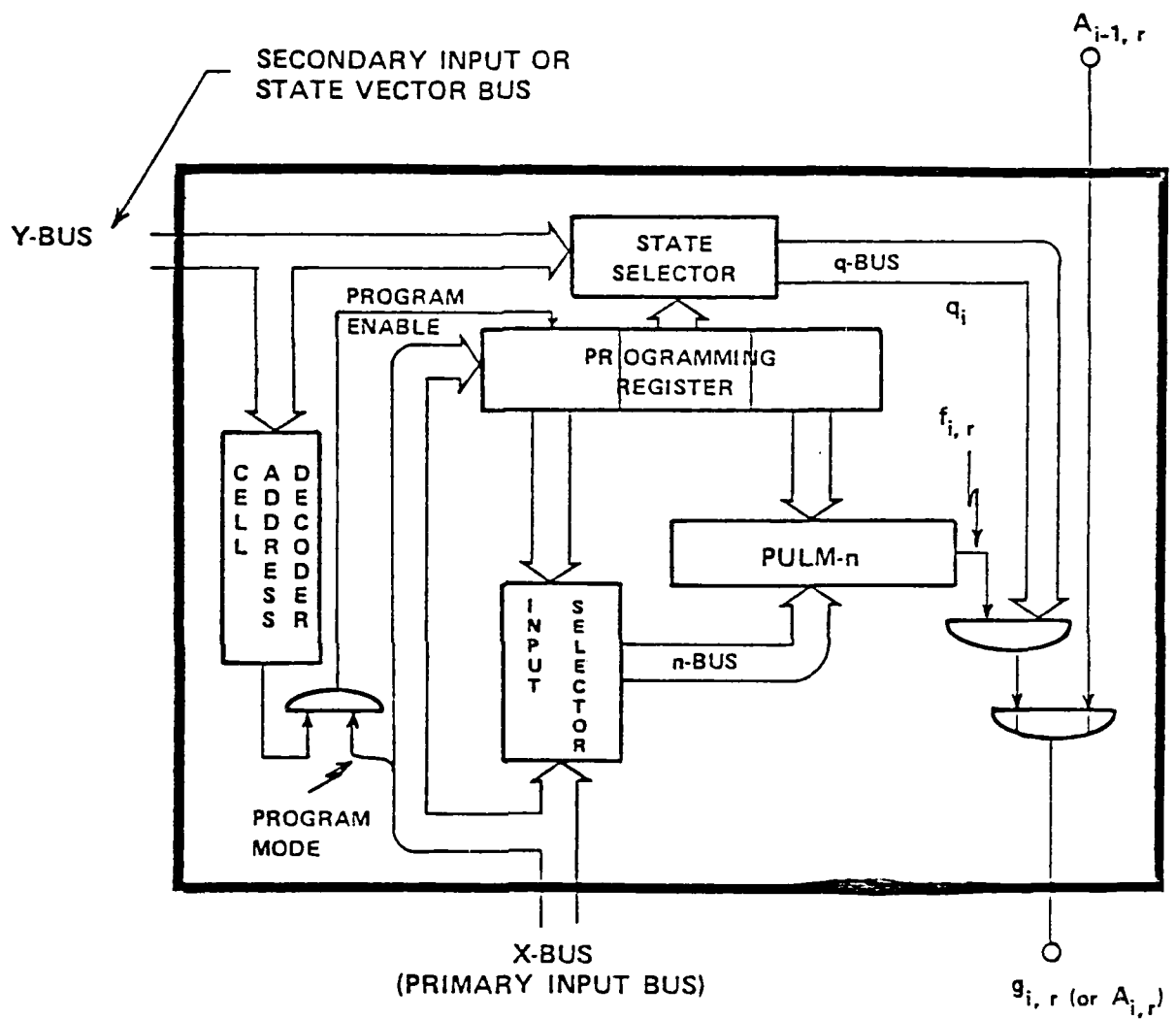


Figure 2.1 Basic Cell

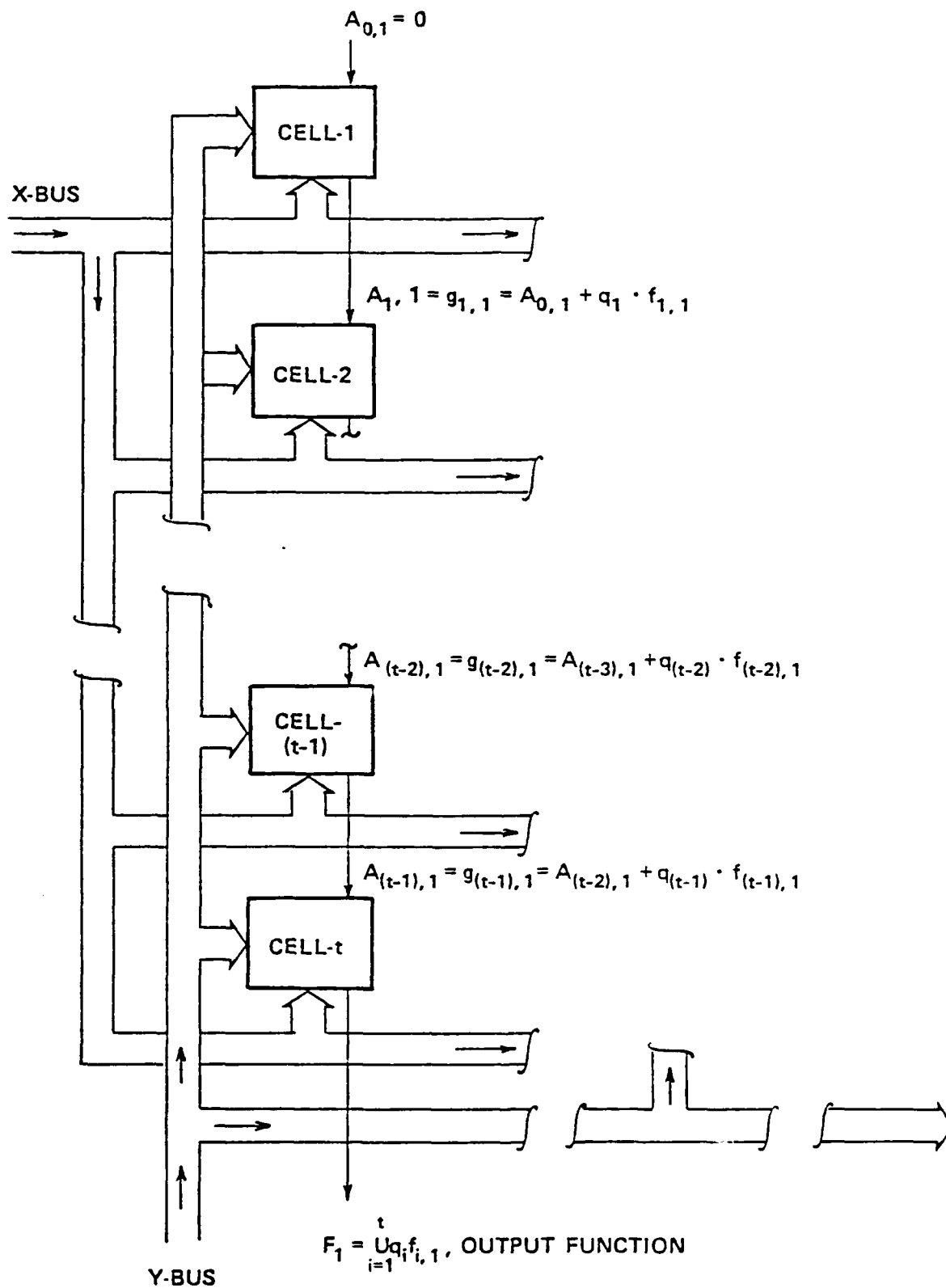


Figure 2.2 A Cellular Cascade

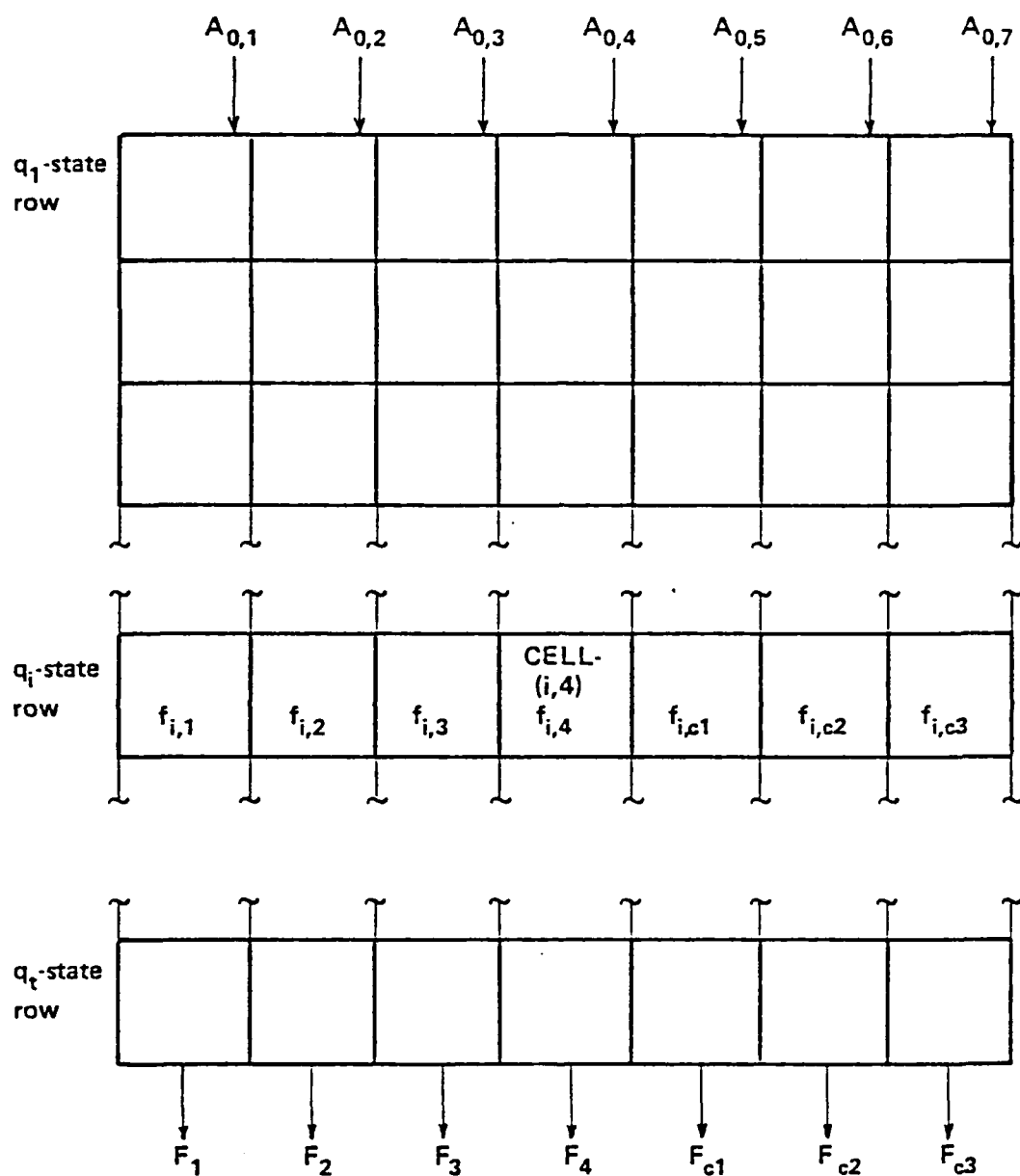


Figure 2.3 Encoded Cellular Array

3.0 DESCRIPTION OF BASIC CELL

The major module of the basic cell shown in Figure 2.1 is the so-called programmable universal logic module (PULM-n) capable of realizing any switching function of n binary variables. The programmability of this module is made possible through an appropriate field in the programming register, while the n binary variables used by the module to form the desired n -variable switching function are provided via an input selector and accessed through the n -BUS under the control of the programming register. Finally, a third field in the programming register is used to select the state q_i associated with the cell in question.

The Input-Selector and State-Selector units receive primary input and secondary input (or state) information via the X-BUS and Y-BUS, respectively. These two busses may be arbitrarily large in size while the q -BUS and n -BUS, referred to earlier, are, for practical considerations, of significantly smaller size. In the case of the n -BUS, this is motivated by the growth in size of the PULM- n unit as n becomes large; with respect to the q -BUS, one is interested in satisfying the state-space of a machine with the minimum number of state variables. For K -out-of- m codes used here, the choice of K , which denotes the size of the q -BUS, must be such that it produces the largest possible number of states from the m state variables brought in via the Y-BUS. Thus, the usual choice for K is such that the expression $\frac{m!}{k!(k-m)!}$ is maximized. It should be noted that with the X-BUS there is an extra line known as the "program mode" line used for controlling the programming register, and its specific function will be revealed shortly.

A unique feature of the proposed basic cell is the dual function served by the X- and Y-BUSSES. In addition to their main function of importing primary and state information to the cell, they also serve two additional functions: The X-BUSS is used to program the programming register, provided, of course, the cell in question has been properly addressed via the Y-BUS to generate the required "program enable" control signal; and second, the Y-BUS is the one responsible for addressing uniquely the cell of interest by making use of a unique "cell address decoder". The dual functions served by the X- and Y-BUSSES constitute a very important design consideration since they help maintain a

low pin-count for the cell. In the "program mode" the cell output is disregarded.

Another important feature of the cell in Figure 2.1 is the fact that all the incoming information is reduced or "fully decoded" to a single-line output which is very important from the standpoint of pin-count.

4.0 DESCRIPTION OF CELLULAR ARRAY

An aggregate of basic cells as arranged in Figure 2.2 forms a cellular array. Each cell has full access to the X- and Y-BUS, and function realization is effected along a column. The upper boundaries of the array are set to zero, and the realized function along each column is output as F_i and represents either an excitation or output function.

Each cell in a column is programmed to account for a specific term of the function which is implemented in a sum-of-products form. For functions which are independent of state information, the "state selector" in each cell is capable of providing the Boolean constant 1 under control from the appropriate field of the programming register, thus facilitating the generation of product terms not requiring state information.

The cellular array described in this study utilizes a basic cell which is functionally more complex than the one used by Page and Marinos [7], and, as a result, it leads to more flexible cellular structures. It should be noted that the proposed cellular array permits localized selection of both state variables and primary input variables and facilitates its own programmability via the primary and secondary input busses X and Y. Each cell in the array is individually addressed, and the array is assumed to obtain its state information from a separate memory array. In the sequel, a functionally improved basic cell is presented which allows intercellular information transfer from each cell to all of its four neighbors, thus overcoming certain limitations of the one-directional transfer of information of the present cellular array. This improved cell results in cellular arrays over which one may invoke optimization procedures resulting in more efficient utilization of the cellular structure.

4.1 A Procedure For Mapping An Arbitrary Synchronous Sequential Machine Into A Cellular Array

The algorithmic procedure which follows is based on a similar procedure given by Page and Marinos [7], and Page [11] and it is formulated in a way which makes its mechanization easy to carry out. A programmed version of the algorithm is available and typical illustrations making use of the program are given in Appendix 1.

Using symbolism previously defined, one may present the algorithm as a five-step procedure.

Step 1

Obtain the so-called connection matrix of the sequential machine in the form

$$C = \begin{matrix} & \begin{matrix} 1 & \dots & j & \dots & t \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ i \\ \vdots \\ t \end{matrix} & \begin{bmatrix} C_{11} & \dots & C_{1t} \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ C_{t1} & \dots & C_{tt} \end{bmatrix} \end{matrix} \begin{matrix} \leftarrow \text{Next State} \\ \\ \\ \leftarrow \text{Present State} \end{matrix}$$

where

$$C_{ij} = \bigcup_h x_h \mid \delta(x_h, q_i) = q_j$$

The entry C_{ij} denotes the union function of all the input symbols x_h which cause a transition from state q_i to state q_j .

Step 2

Obtain the so-called output matrix in the form

$$W = \begin{matrix} & \begin{matrix} 1 & \dots & s & \dots & p \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ i \\ \vdots \\ t \end{matrix} & \begin{bmatrix} a_{11} & \dots & a_{1s} & \dots & a_{1p} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{t1} & \dots & a_{ts} & \dots & a_{tp} \end{bmatrix} \end{matrix} \begin{matrix} \leftarrow \text{Primary Output Lines} \\ \\ \\ \leftarrow \text{Present State} \end{matrix}$$

where

$$a_{is} = w_s(q_i) \quad (\text{Moore-Machine})$$

and

$$a_{is} = \bigcup_h x_h \mid w_s(x_h, q_i) = 1 \quad (\text{Mealy-Machine})$$

The entry a_{is} denotes, in a Moore-Machine, the binary value of the output line "s" when the machine is in state q_i . In the case of a Mealy-Machine, the entry a_{is} represents the union function of all the input symbols which cause transitions from state q_i to other states resulting in an output pulse on line s. It should be pointed out that the above formulation allows some of the

output lines to display Mealy behavior and the remaining Moore type behavior in sequential machine design in which both types of output behavior are present.

Step 3

(This step is used only to facilitate the theoretical discussion leading to the development of certain rules, and not for the purpose of actually obtaining matrix M_E). Consider the so-called memory excitation matrix

$$M_E = \begin{matrix} & \begin{matrix} 1 & \dots & j & \dots & r \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ i \\ \vdots \\ t \end{matrix} & \begin{bmatrix} m_{11} & \dots & m_{1j} & \dots & m_{1r} \\ \vdots & & \vdots & & \vdots \\ \vdots & & m_{ij} & & \vdots \\ \vdots & & \vdots & & \vdots \\ m_{t1} & \dots & m_{tr} \end{bmatrix} \end{matrix} \begin{matrix} \leftarrow \text{Memory Element} \\ \\ \uparrow \text{Present State} \end{matrix}$$

where the entry m_{ij} is a functional form dependent on the type of memory element used, and analytically expresses the relevant excitation term of the j th memory element when in state q_i , that is

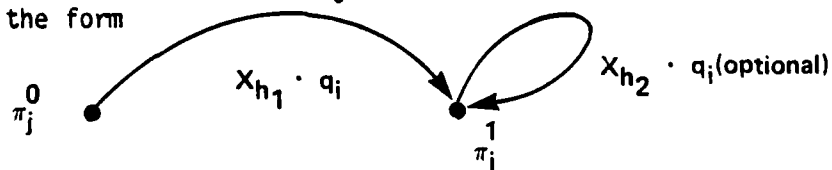
$$m_{ij} = f_{ij}(x_1, x_2, \dots, x_n) \cdot q_i$$

a) Linear delay memory

For linear delay memory element, the entry m_{ij} is given by

$$m_{ij} = \bigcup_h x_h q_i \quad \left| \quad \delta(x_h, q_i) \in \pi_j^1 \right.$$

where π_j^1 denotes the partition block of the state space identified with the uncomplemented version of the j th state variable. Thus, one may identify the primary inputs x_h present in entry m_{ij} as those responsible for transitions of the form



(out of state q_i) as far as state variable y_j is concerned.

b) Trigger flip-flop memory

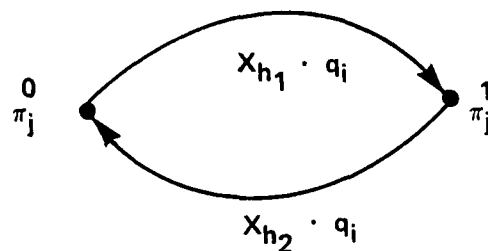
For trigger flip-flops, the entry m_{ij} is given by

$$m_{ij} = \bigcup_h x_h \cdot q_i \quad \left| \begin{array}{l} q_i \in \pi_j^0 \rightarrow \delta(x_h, q_i) \in \pi_j^1 \\ \text{or} \\ q_i \in \pi_j^1 \rightarrow \delta(x_h, q_i) \in \pi_j^0 \end{array} \right.$$

Here, the entry m_{ij} is the union function of those primary inputs x_h which are responsible for causing transitions from state $q_i \in \pi_j^1$ to states

$$\delta(x_h, q_i) \in \pi_j^1 \text{ or vice versa}$$

Symbolically, such transitions are of the form



(out of state q_i) as far as state variable y_j is concerned.

c) Set-Reset flip-flop memory

For set-reset flip-flops, the entry m_{ij} is given by

$$m_{ij} = [m_{ij}(S), m_{ij}(R)]$$

where $m_{ij}(S)$ is the "SET" component, and $m_{ij}(R)$ is the "RESET" component.

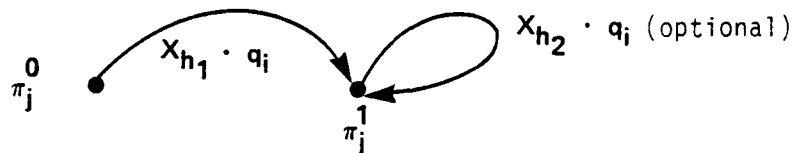
$$m_{ij}(S) = \bigcup_h x_h \cdot q_i \quad \left| \begin{array}{l} q_i \in \pi_j^0 \rightarrow \delta(x_h, q_i) \in \pi_j^1; \\ \text{the } x_h \text{'s for which} \\ q_i \in \pi_j^1 \rightarrow \delta(x_h, q_i) \in \pi_j^0 \\ \text{are to be included in the} \\ \text{union function optionally.} \end{array} \right.$$

The entry $m_{ij}(S)$ is the union function comprised of those primary inputs x_h which are responsible for transitions from state $q_i \in \pi_j^0$ to states

$$\delta(x_h, q_i) \in \pi_j^1,$$

necessarily; and optionally of those x_h such that

for $q_i \in \pi_j^1$ one has $\delta(x_h, q_i) \in \pi_j^1$. Symbolically, such transitions are of the form

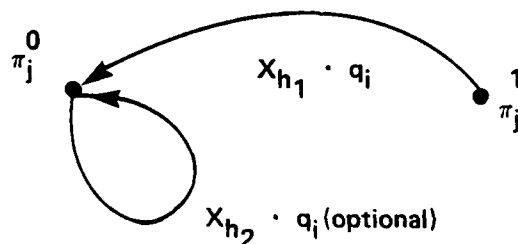


(out of state q_i) as far as state variable y_j is concerned.

Similarly, for the "RESET" function $m_{ij}(R)$, one has

$$m_{ij}(R) = \bigcup_h x_h \cdot q_i \quad \left| \begin{array}{l} q_i \in \pi_j^1 \rightarrow \delta(x_h, q_i) \in \pi_j^0; \\ \text{the } x_h \text{'s for which} \\ q_i \in \pi_j^0 \rightarrow \delta(x_h, q_i) \in \pi_j^0 \\ \text{are to be included in the} \\ \text{union function, optionally} \end{array} \right.$$

The entry $m_{ij}(R)$ may be described in a way analogous to the $m_{ij}(S)$ function, and symbolically, those transitions of the state variable y_j which contribute to the function $m_{ij}(R)$ are of the form



d) J-K flip-flop memory

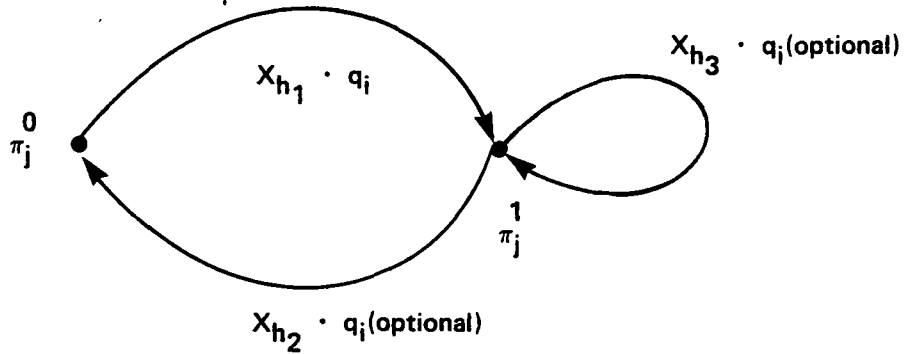
For J-K flip-flops, the entry m_{ij} is a composite entry given by

$$m_{ij} = [m_{ij}(J), m_{ij}(K)]$$

where

$$m_{ij}(J) = \bigcup_h x_h \cdot q_i \quad \left| \begin{array}{l} q_i \in \pi_j^0 \rightarrow \delta(x_h, q_i) \in \pi_j^1; \\ \text{the } x_h \text{'s for which either} \\ q_i \in \pi_j^1 \rightarrow \delta(x_h, q_i) \in \pi_j^1 \\ \text{or } q_i \in \pi_j^0 \rightarrow \delta(x_h, q_i) \in \pi_j^0 \\ \text{are to be included in the} \\ \text{union function, optionally} \end{array} \right.$$

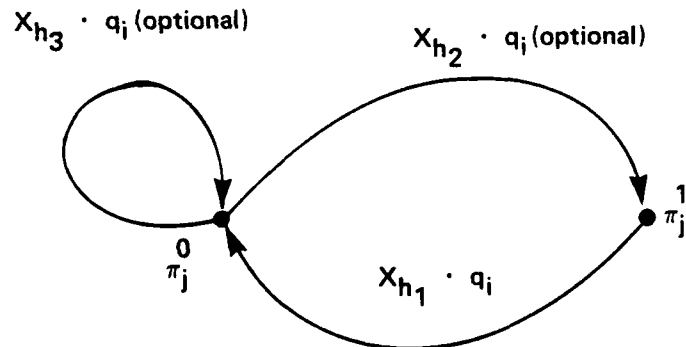
The description for the $m_{ij}(J)$ function is similar to the description of $m_{ij}(S)$. Symbolically, one is interested in x_h 's responsible for transitions (out of state q_i) of the form



Similarly, for $m_{ij}(K)$ one has

$$m_{ij}(K) = \bigcup_h x_h \cdot q_i \left\{ \begin{array}{l} q_i \in \pi_j^1 \xrightarrow{1} \delta(x_h, q_i) \in \pi_j^0; \\ \text{the } x_h \text{'s for which} \\ \text{either } q_i \in \pi_j^0 \xrightarrow{0} \delta(x_h, q_i) \in \pi_j^1 \\ \text{or } q_i \in \pi_j^1 \xrightarrow{1} \delta(x_h, q_i) \in \pi_j^0 \\ \text{are to be included in the} \\ \text{union function optionally.} \end{array} \right.$$

Symbolically, one considers only x_h 's responsible for transitions (out of q_i) of the form



The above formulations provide, in effect, the rules through which one may now proceed in a systematic way to obtain the excitation functions for the memory elements, utilizing, of course, the information contained in the connection matrix. Towards this end, one must assign k-out-of-r type codes to all machine states and identify all the states in which the j th state variable (i.e., state variable y_j) is present (i.e., $y_j = 1$) and associate them with the state partition block π_j^1 , and the remaining states with partition block π_j^0 .

In order to facilitate the mechanization of the algorithmic process, we make use of the transpose matrix C which assumes the form

$$C^T = \begin{matrix} & \begin{matrix} 1 & \dots & i & \dots & t \end{matrix} \\ \begin{matrix} 1 \\ \vdots \\ j \\ \vdots \\ t \end{matrix} & \begin{bmatrix} C_{11} & \dots & \dots & \dots & C_{t1} \\ \vdots & \dots & C_{ij} & \dots & \vdots \\ C_{1t} & \dots & \dots & \dots & C_{tt} \end{bmatrix} \end{matrix}$$

← Present State

↑ Next State

and then proceed to generate the matrix

$$C^* = C^T * \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_t \end{bmatrix} = C^T * Q$$

where the $*$ denotes matrix "multiplication" in an AND-like sense ANDing an entry of C^T with a corresponding entry of Q . Matrix C^* is in effect a form of "next-state" matrix of the system, and it will be utilized in a way which leads to a set of elemental excitation matrices from which by direct application of the symbolic rules developed for the various memory elements, earlier, one may obtain the corresponding excitation functions.

Step 4

The elemental excitation matrix is associated with a state variable, say, y_j , and has the form

$$E_j = \begin{matrix} & \begin{matrix} \pi_j^0 & \pi_j^1 \end{matrix} \\ \begin{matrix} \pi_j^0 \\ \pi_j^1 \end{matrix} & \begin{bmatrix} e_{00j} & e_{10j} \\ e_{01j} & e_{11j} \end{bmatrix} \end{matrix}$$

← Present Partition Block

↑ Next Partition Block

where the entry e_{pqj} , $p, q \in [0,1]$, denotes the excitation required by state variable y_j to perform the transition from state p to state q . More specifically, e_{10j} is the excitation which, if applied when $y_j = 1$, will cause a change in the state of y_j to $y_j = 0$.

In order to obtain the elemental excitation matrices, we must first produce the encoded state vector

$$G(r,k) = \begin{bmatrix} G_1(r,k) \\ G_2(r,k) \\ \vdots \\ G_t(r,k) \end{bmatrix}$$

where

$$G_i(r,k) = q_i = y_{i_1} y_{i_2} \dots y_{i_k}$$

with r denoting the number of available state variables and $i = 1, 2, \dots, t$. Any other arbitrary state encoding could have been used just as well. For analytical convenience, the entries of $G(r,k)$ will be replaced by their corresponding k -out-of- r codes. Thus, one may view $G(r,k)$ as a binary matrix of t rows and r columns with the rows denoting states and the columns being identified with the r state variables. Multiplying next, in an AND-OR sense, matrix C^* with $G(r,k)$ and also $\bar{G}(r,k)$, the complement of $G(r,k)$, one has two new matrices, namely,

$$D = C^* \otimes \bar{G}(r,k)$$

and

$$D_1 = C^* \otimes G(r,k)$$

where \otimes denotes AND-OR matrix multiplication D_0 and D_1 contain all the information needed to obtain the elemental excitation matrices. More specifically, D_0 provides the excitations required by state variable y_j , being presently in any of the states in which $y_j = 0$, to assume its uniquely assigned next-state value. Similarly, D_1 provides the excitations required by state variable y_j , being presently in any of the states in which $y_j = 1$, to

assume its uniquely assigned next-state value. To obtain, next, the elemental excitation matrices, E_j , we form the matrices

$$H_{Rj} \text{ and } H_{Cj}$$

where H_{Rj} is a two-row matrix in which the first row is the j th column of $\bar{G}(r,k)$ transposed and the second row is the j th column of $G(r,k)$ transposed. In the case of H_{Cj} , this is a two-column matrix formed by taking the j th column of D_0 first and the j th column of D_1 , secondly. The elemental excitation matrix is now given by

$$E_j = H_{Rj} \otimes H_{Cj}$$

where

$$j = 1, 2, \dots, r$$

The derivation of the elemental excitation matrices is independent of the type of memory element used and it is only through application of previously developed rules that one may interpret the entries of E_j for obtaining the specific excitation functions required by the respective memory element, j . In the case of a linear delay element, j , the appropriate rule states that its excitation, D_j , is given by

$$D_j = e_{01j} + e_{11j}$$

For a trigger flip-flop, the rules dictate the excitation

$$T_j = e_{01j} + e_{10j}$$

For the Set-Reset flip-flop, the rules result in a set-function

$$S_j = e_{01j} + e_{11j}$$

with the term e_{11j} used optionally; and a reset-function

$$R_j = e_{10j} + e_{00j}$$

with the term e_{00j} used optionally. Finally, for the J-K flip-flop, the pertinent rules yield

$$J_j = e_{01j} + (e_{10j} + e_{11j}),$$

with the term $(e_{10j} + e_{11j})$ used optionally; and

$$K_j = e_{10j} + (e_{01j} + e_{00j}),$$

with the term $(e_{01j} + e_{00j})$ used optionally. Having obtained in this manner the excitation functions for all memory elements j one has, in effect, determined the entries of matrix M_E since the union function

$$U_{ij}^t = \bigcup_{i=1}^t f_{ij}^t(x_1, x_2, \dots, x_n) q_i$$

is by definition the excitation function of the j th memory element.

Step 5

Obtain the output function matrix

$$Z = \begin{bmatrix} z_1 \\ \vdots \\ z_j \\ \vdots \\ z_p \end{bmatrix}$$

where z_p denotes the j th output function. As stated previously, the output may be either of the Mealy or the Moore type or even a combination of the two. In any case, the Z -matrix is easily obtained as

$$Z = W^T \otimes G(r, k)$$

where W^T is the transpose of W and $G(r, k)$ denotes the state encoding vector as defined earlier. That is,

$$G(r,k) = \begin{bmatrix} y_{1_1} & y_{1_2} & \dots & y_{1_k} \\ \vdots & \vdots & & \vdots \\ y_{i_1} & y_{i_2} & \dots & y_{i_k} \\ \vdots & \vdots & & \vdots \\ y_{t_1} & y_{t_2} & \dots & y_{t_k} \end{bmatrix}$$

As it was stated previously, the choice of state encoding is quite arbitrary with the selection of $G(r,k)$, in this case, favored by various design considerations dictated by the cellular structure under consideration.

The above procedure is quite general and easily programmed to synthesize any arbitrary synchronous sequential machine through the use of matrix manipulations based on the connection and output matrices of the sequential machine.

4.2 Illustration of Mapping Procedure

The following state transition table describes the sequential machine to be embedded into the cellular array discussed earlier.

Table 4.1 Machine State Transition

Present State	INPUT		Level Output, z
	x_1	x_2	
q_1	$q_1/1$	$q_5/0$	0
q_2	$q_1/0$	$q_3/0$	1
q_3	$q_2/0$	$q_4/1$	0
q_4	$q_3/1$	$q_3/0$	0
q_5	$q_3/0$	$q_4/0$	1

Next State/Output

Step 1

From Figure 4.1 one obtains the corresponding connection matrix

$$C = \begin{bmatrix} x_1 & 0 & 0 & 0 & x_2 \\ x_1 & 0 & x_2 & 0 & 0 \\ 0 & x_1 & 0 & x_2 & 0 \\ 0 & 0 & x_1+x_2 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 \end{bmatrix}$$

Step 2

From Figure 4 one may also obtain the corresponding output matrix

$$W = \begin{bmatrix} x_1 & 0 \\ 0 & 1 \\ x_2 & 0 \\ x_1 & 0 \\ 0 & 1 \end{bmatrix}$$

noting that the sequential network has a pulse output line and a level output line.

Step 3

Obtain the C^* matrix

$$\begin{aligned} C^* &= C^T * Q \text{ or} \\ C^* &= \begin{bmatrix} x_1 & x_1 & 0 & 0 & 0 \\ 0 & 0 & x_1 & 0 & 0 \\ 0 & x_2 & 0 & x_1+x_2 & x_1 \\ 0 & 0 & x_2 & 0 & x_2 \\ x_2 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix} \\ &= \begin{bmatrix} x_1 q_1 & x_1 q_2 & 0 & 0 & 0 \\ 0 & 0 & x_1 q_3 & 0 & 0 \\ 0 & x_2 q_2 & 0 & (x_1+x_2) q_4 & x_1 q_5 \\ 0 & 0 & x_2 q_3 & 0 & x_2 q_5 \\ x_2 q_1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

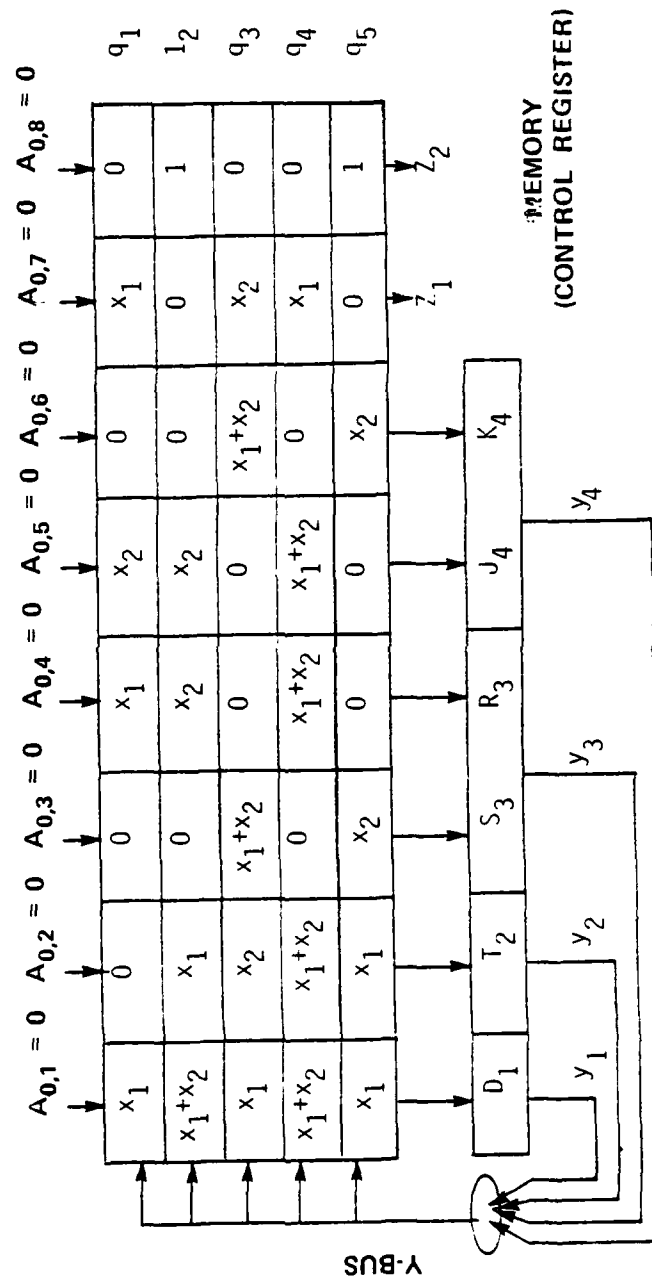


Figure 4.1. Example Array

Step 4

In order to obtain the elemental excitation matrices, we must first decide on a specific state encoding which, in this case, we assume to be a "2-out-of-4" with codes assigned to states as follows:

$$G(4,2) = \begin{bmatrix} y_1 & y_2 \\ y_1 & y_3 \\ y_1 & y_4 \\ y_2 & y_3 \\ y_2 & y_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix}$$

The corresponding binary form of matrix $G(4,2)$ is

$$G(4,2) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Next we construct the auxiliary matrices, using the binary form of $G(4,2)$,

$$D_0 = C * \bar{\otimes} G(4,2)$$

and

$$D_1 = C * \otimes G(4,2);$$

that is

$$D_0 = \begin{bmatrix} 0 & x_1 q_2 & x_1 q_1 & x_1 q_1 + x_1 q_2 \\ 0 & x_1 q_3 & x_1 q_3 & 0 \\ (x_1 + x_2) q_4 + x_1 q_5 & x_2 q_2 & x_1 q_5 & x_2 q_2 + (x_1 + x_2) q_4 \\ x_2 q_5 & x_2 q_3 & x_2 q_3 + x_2 q_5 & 0 \\ 0 & 0 & x_2 q_1 & x_2 q_1 \end{bmatrix}$$

and

$$D_1 = \begin{bmatrix} x_1 q_1 + x_1 q_2 & x_1 q_1 & x_1 q_2 & 0 \\ x_1 q_3 & 0 & 0 & x_1 q_3 \\ x_2 q_2 & (x_1 + x_2) q_4 + x_1 q_5 & x_2 q_2 + (x_1 + x_2) q_4 & x_1 q_5 \\ x_2 q_3 & x_2 q_5 & 0 & x_2 q_3 + x_2 q_5 \\ x_2 q_1 & x_2 q_1 & 0 & 0 \end{bmatrix}$$

Next, form the second set of auxiliary matrices H_{Rj} and H_{Cj} , where $j=1,2,3,4$. That is,

$$H_{R1} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad H_{C1} = \begin{bmatrix} 0 & x_1 q_1 + x_1 q_2 \\ 0 & x_1 q_3 \\ (x_1 + x_2) q_4 + x_1 q_5 & x_2 q_2 \\ x_2 q_5 & x_2 q_3 \\ 0 & x_2 q_1 \end{bmatrix}$$

$$H_{R2} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad H_{C2} = \begin{bmatrix} x_1 q_2 & x_1 q_1 \\ x_1 q_3 & 0 \\ x_2 q_2 & (x_1 + x_2) q_4 + x_1 q_5 \\ x_2 q_3 & x_2 q_5 \\ 0 & x_2 q_1 \end{bmatrix}$$

$$H_{R3} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \quad H_{C3} = \begin{bmatrix} x_1 q_1 & x_1 q_2 \\ x_1 q_3 & 0 \\ x_1 q_5 & x_2 q_2 + (x_1 + x_2) q_4 \\ x_2 q_3 + x_2 q_5 & 0 \\ x_2 q_1 & 0 \end{bmatrix}$$

$$H_{R4} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}, \quad H_{C4} = \begin{bmatrix} x_1 q_1 + x_1 q_2 & 0 \\ 0 & x_1 q_3 \\ x_2 q_2 + (x_1 + x_2) q_4 & x_1 q_5 \\ 0 & x_2 q_3 + x_2 q_5 \\ x_2 q_1 & 0 \end{bmatrix}$$

From the above auxiliary matrices, one may now obtain the elemental excitation matrices E_j , where $j=1,2,3,4$. They are,

$$E_1 = H_{R1} \otimes H_{C1} = \begin{bmatrix} x_2 q_5 & x_2 q_3 + x_2 q_1 \\ (x_1 + x_2) q_4 + x_1 q_5 & x_1 q_1 + (x_1 + x_2) q_2 + x_1 q_3 \end{bmatrix}$$

$$E_2 = H_{R2} \otimes H_{C2} = \begin{bmatrix} x_1 q_3 + x_2 q_2 & (x_1 + x_2) q_4 + x_1 q_5 \\ x_1 q_2 + x_2 q_3 & (x_1 + x_2) q_1 + x_2 q_5 \end{bmatrix}$$

$$E_3 = H_{R3} \otimes H_{C3} = \begin{bmatrix} (x_1 + x_2) q_1 + x_1 q_5 & x_1 q_2 + x_2 q_2 + (x_1 + x_2) q_4 \\ (x_1 + x_2) q_3 + x_2 q_5 & 0 \end{bmatrix}$$

$$E_4 = H_{R4} \otimes H_{C4} = \begin{bmatrix} x_1 q_1 + x_1 q_2 & (x_1 + x_2) q_3 + x_2 q_5 \\ x_2 q_1 + x_2 q_2 + (x_1 + x_2) q_4 & x_1 q_5 \end{bmatrix}$$

Noting now that

$$E_j = \begin{bmatrix} e_{00j} & e_{10j} \\ e_{01j} & e_{11j} \end{bmatrix}$$

and assuming, for pedagogical purposes, the first element to be a linear delay; the second, a trigger flip-flop; the third, a set-reset flip-flop; and the fourth, a J-K flip-flop. One may interpret the four elemental excitation matrices correspondingly using appropriately developed rules to obtain

$$D_1 = e_{011} + e_{111} \quad \text{or}$$

$$D_1 = (x_1 + x_2) q_4 + x_1 q_5 + x_1 q_1 + (x_1 + x_2) q_2 + x_1 q_3$$

$$T_2 = e_{012} + e_{102} \quad \text{or}$$

$$T_2 = x_1 q_2 + x_2 q_3 + (x_1 + x_2) q_4 + x_1 q_5$$

$$S_3 = e_{013} + e_{113}$$

$$R_3 = e_{103} + e_{003} \quad \text{or}$$

$$S_3 = (x_1 + x_2) q_3 + x_2 q_5$$

$$R_3 = x_1 q_1 + x_2 q_2 + (x_1 + x_2) q_4 + \underbrace{\{(x_1 + x_2) q_1 + x_1 q_5\}}_{\text{Optional Term}}$$

$$J_4 = e_{014} + (e_{104} + e_{114})$$

$$K_4 = e_{104} + (e_{014} + e_{004}) \quad \text{or}$$

$$J_4 = x_2 q_1 + x_2 q_2 + (x_1 + x_2) q_4 + \underbrace{\{(x_1 + x_2) q_3 + x_2 q_5 + x_1 q_5\}}_{\text{Optional Term}}$$

$$K_4 = (x_1 + x_2) q_3 + x_2 q_5 + \underbrace{\{x_2 q_1 + x_2 q_2 + (x_1 + x_2) q_4 + x_1 q_1\}}_{\text{Optional Term}}$$

Step 5

In order to obtain the output function matrix, we make use of matrix W and G(r,k); that is,

$$Z = W \otimes G(r,k)$$

$$= \begin{bmatrix} x_1 & 0 & x_2 & x_1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} y_1 & y_2 \\ y_1 & y_3 \\ y_1 & y_4 \\ y_2 & y_3 \\ y_2 & y_4 \end{bmatrix}$$

or

$$z_1 = z_{\text{Pulse}} = x_1 y_1 y_2 + x_2 y_1 y_4 + x_1 y_2 y_3$$

$$z_2 = z_{\text{Level}} = y_1 y_3 + y_2 y_4$$

or equivalently,

$$z_1 = x_1 q_1 + x_2 q_3 + x_1 q_4$$

$$z_2 = q_2 + q_5$$

Using next the cellular cascade of Figure 2.2, the array form of Figure 2.3, and assuming the postulated memory elements, one obtains the cellular array implementation shown in Figure 4.1 for the above example. In this example only the Y-BUS is shown with the X-BUS implied and each row of the array associated only with one state as designated on the right side of the array. This need not be so since each cell permits the realization of any state code.

The array implementation in Figure 4.1 requires $5 \times 8 = 40$ cells of which only 24 are actually used. The unused array capacity may be utilized as spare or standby resource to be used for system repair or possibly for "hardware encoding" the array for purposes of fault-detection and fault-correction. Approaches are presently investigated for reducing as well as optimizing in certain ways the location of unused cells (i.e., zero entries) in the array, and the results of this work will be reported in the near future. Methods of "hardware encoding" arrays, such as the one in Figure 4.1, will be discussed in the sequel.

4.3 Cellular Arrays with Hardware Encoded BIT Facilities

The term "hardware encoding" is used to mean the process of mapping, in a systematic way, any of the well-known information encoding schemes, used for error control, into the hardware structure of a cellular array.

The manner in which such a mapping is carried out is greatly simplified by assuming that all the cells of a row are programmed to receive the same state code. This is only a topological and not a functional restriction imposed on the machine which is embedded in the cellular array and it is made for the sake of analytical convenience. Thus, the mapping of a sequential machine into a cellular array, using the algorithm presented earlier, results in a cellular machine structure in which each row is uniquely associated with a machine state, except in the case of state splitting to be discussed later.

Figure 2.3 illustrates a cellular array in which the i^{th} row is associated with machine state q_i . The busses and other details of the cellular array are omitted for the sake of clarity. The four leftmost columns of the array are used to implement machine functions (i.e., excitation and output functions) while the remaining three (i.e., F_{c1} , F_{c2} and F_{c3}) denote the built-in check functions employed, in this case, to encode the information carrying functions F_1 , F_2 , F_3 and F_4 according to the well-known single-error correction Hamming code. These checking functions are implemented in the last three columns of the array in a manner similar to that used to form any other machine function. The resulting array is one with "hardware encoded" BIT facilities that make continuous and non-interfering monitoring of the encoded output (i.e., $F_1 F_2 F_3 F_4 F_{c1} F_{c2} F_{c3}$), using standard error checking schemes, possible.

Although the illustration in Figure 2.3 uses a single-error-correcting Hamming code, other encoding schemes may be just as easily employed by utilizing the appropriate relations which must hold between the "information cell" function $f_{i,r}$'s and the "checking cell" functions $f_{i,cj}$'s. For the single-error-correcting Hamming code used in Figure 2.3, the well-known relationships are given by the equations

$$\begin{aligned} f_{i,c1} &= f_{i,1} \oplus f_{i,2} \oplus f_{i,4} \\ f_{i,c2} &= f_{i,1} \oplus f_{i,3} \oplus f_{i,4} \\ f_{i,c3} &= f_{i,2} \oplus f_{i,3} \oplus f_{i,4} \end{aligned} \tag{4}$$

assuming, of course, even parity. The above equations prescribe the respective "checking cell" functions that must be programmed so that when the machine is in state q_r , the r^{th} row of the array is "hardware encoded" in accordance with the coding scheme of interest.

In the case of a combinational (i.e., memoryless) network, the checking functions are realized in a manner similar to the sequential network case by assuming the network as being a one-state machine. Whenever the functional complexity of a machine exceeds the resources of a row in a cellular array, then more than one row may be associated with a single state resulting in what we have previously referred to as state splitting.

The approach outlined above for realizing BIT facilities is compatible with LSI technologies, and it makes full utilization of well-known information encoding schemes used for error detection and diagnosis in digital systems. It is worth noting that "hardware encoding" BIT facilities, as proposed here, do not require specially designed hardware, and they are incorporated into the overall system in a way that makes them a natural extension of the non-encoded machine structure.

In the specific case of the example of Figure 4.1, encoding of the 8-BIT information word, comprised of six bits (i.e., six columns) for implementing the excitation functions and two bits (i.e., two columns) for realizing the output functions, according to a single-error correction code results in the array shown in Figure 4.2. This so-called "hardware encoded" cellular array utilizes

four additional columns and, of course, an appropriate Hamming decoder (i.e., checker) monitoring the 12-bit output from the cellular array. The Hamming decoder may be a standard unit of specified word length and several of them could be used for on-line monitoring of cellular arrays of arbitrary size.

Figure 4.2 displays also the use of a check attached to the Y-BUS which is capable of on-line monitoring of the state vector which we assume to be appropriately encoded. In the example of Figure 4.2 it is assumed that the state vector is encoded as "2-out-of-4" code. In this specific instance, the state vector checker may be designed to be a totally self-checkable unit [9].

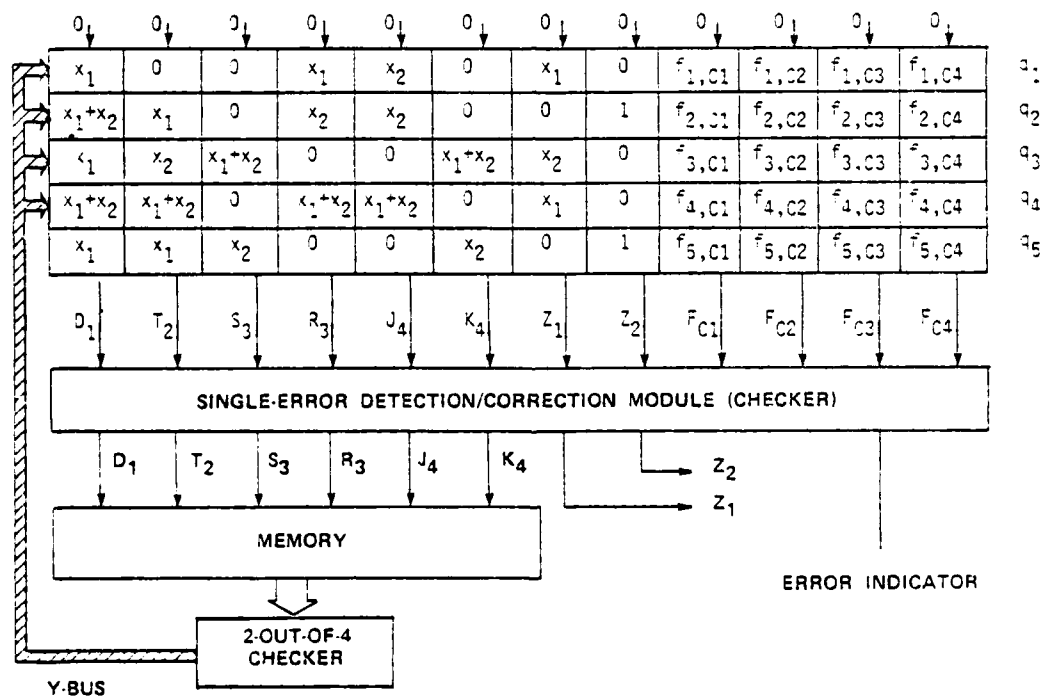


Figure 4.2 Example of "Hardware Encoded" Cellular Array

Using an approach analogous to that employed to obtain the relationships given by Equation (4), one may now show that for even parity and eight information bits the required checking functions to be used for a Hamming single-error correction encoding are:

$$\begin{aligned}
 f_{i,c1} &= f_{i,1} \oplus f_{i,2} \oplus f_{i,4} \oplus f_{i,5} \oplus f_{i,7} \\
 f_{i,c2} &= f_{i,1} \oplus f_{i,3} \oplus f_{i,4} \oplus f_{i,6} \oplus f_{i,7} \\
 f_{i,c3} &= f_{i,2} \oplus f_{i,3} \oplus f_{i,4} \oplus f_{i,8} \\
 f_{i,c4} &= f_{i,5} \oplus f_{i,6} \oplus f_{i,7} \oplus f_{i,8}
 \end{aligned} \tag{5}$$

For purposes of illustrating the use of Equation (5), consider the case $i=3$ as it applies to Figure 4.2. Since

$$\begin{aligned}
 f_{3,1} &= x_1 & f_{3,2} &= x_2 & f_{3,3} &= x_1 + x_2 & f_{3,4} &= 0 \\
 f_{3,5} &= 0 & f_{3,6} &= x_1 + x_2 & f_{3,7} &= x_2 & f_{3,8} &= 0,
 \end{aligned}$$

the resulting expressions are,

$$\begin{aligned}
 f_{3,c1} &= x_1 \oplus x_2 \oplus 0 \oplus 0 \oplus x_2 = x_1 \\
 f_{3,c2} &= x_1 \oplus (x_1 + x_2) \oplus 0 \oplus (x_1 + x_2) \oplus x_2 = x_1 \oplus x_2 \\
 f_{3,c3} &= x_2 \oplus (x_1 + x_2) \oplus 0 \oplus 0 = x_1 \bar{x}_2 \\
 f_{3,c4} &= 0 \oplus (x_1 + x_2) \oplus x_2 \oplus 0 = x_1 \bar{x}_2
 \end{aligned}$$

The remaining entries to be programmed into the checking portion of the cellular array are obtained in a similar manner.

5.0 AN ALTERNATE BASIC CELL DESIGN

It was previously stated that one of the disadvantages of the basic cell described earlier was the fact that it could only pass information to the cell below and thus, in many instances, could result in a large number of array cells not being utilized. Even though this could be viewed as reserve capacity (i.e., spare cells) in the column in which they appear, they are still of no use to other columns where they might be needed, and therefore some means of exercising greater control over their allocation is very much in order.

Figure 5.1 shows an alternate cell design which imparts increased control capability to the cellular structure over individual cells. The difference between this cell and that shown in Figure 5.1 is the fact that its programming register has a fourth field through which, using an appropriate selector, one may select for cell input the output of one of its four immediate neighbors. Furthermore, the output of the cell is broadcast to all of its four immediate neighbors, and it may be selected by any and all of them as their input according to need.

Functionally, both cells are designed around the same logical expressions developed earlier and differ only in intercell communication capabilities. Significant progress towards developing algorithmic procedures which take into account the enhanced intercell communication capability of the new cell has been made and will be reported in the near future.

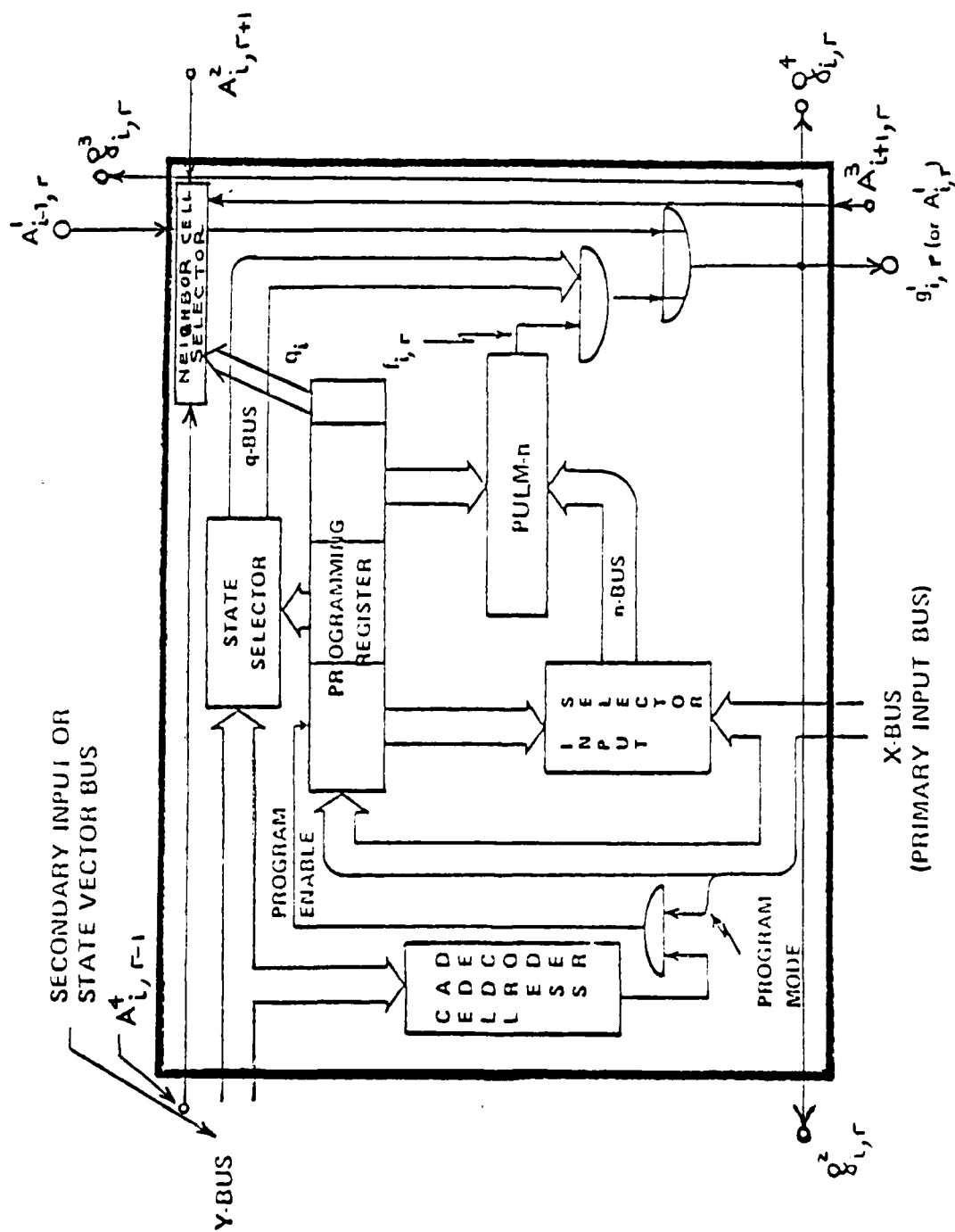


Figure 5.1 Alternate Basic Cell

6.0 CELLULAR ARRAY IMPLEMENTATION OF A MICROPROCESSOR SYSTEM

This section offers a preview of a study which has been undertaken to demonstrate the utility of cellular arrays as a design tool for realizing digital systems with built-in-test facilities. In order to draw a meaningful comparison in terms of system performance, it was decided to utilize a popular LSI chip of significant complexity and have it implemented in a cellular array structure.

The LSI processor chosen is the Intel 8080, and the mapping of its hardware organization on the cellular array, as presently planned, follows the functional array segmentation shown in Figure 6.1. The choice of microprocessor chip is not viewed as critical, and its selection in this case was based primarily on our greater familiarity with the Intel 8080 over other possible choices. It should be pointed out that in Figure 6.1 omitted memory and register facilities.

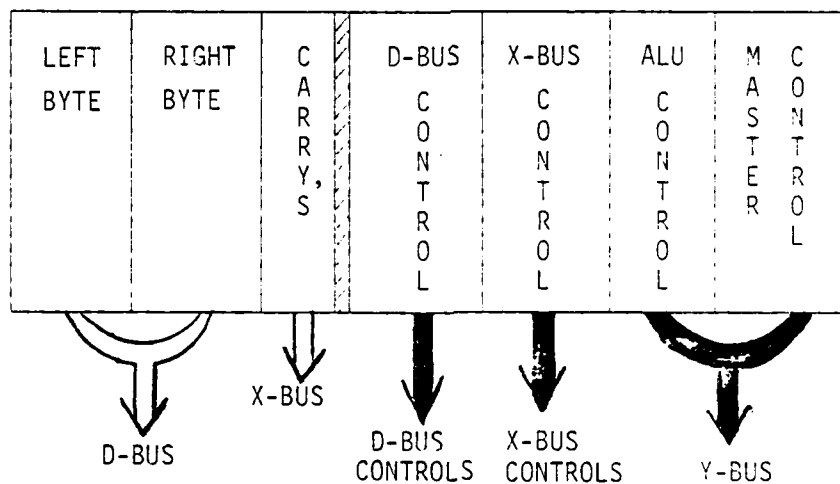


Figure 6.1 A Proposed Cellular Array Organization for a Microprocessor Unit.

A brief functional description of the various array segments shown in Figure 6.1 follows:

ALU-unit: The ALU-unit performs the arithmetic and logic operations

such as ADD, INCREMENT, XOR, etc. The Y-BUS information from the "ALU Control" unit determines the operation to be performed on the data transferred from designated registers via the X-BUSS and the CARRY'S output goes directly to the X-BUS so that a look-ahead of 4 is utilized.

D-BUS control unit: It generates signals for the control of information flow from ALU to various registers.

X-BUS control unit: It generates signals for the control of information flow from various registers to the X-BUS.

ALU-Control unit: It generates and broadcasts control signals to the ALU unit via the Y-BUS.

Master-Control unit: It generates internal machine states and determines the machine cycle to be carried out based on op-code and present state information.

The BIT facilities in the array of Figure 6.1 are implicit and the total width of the cellular structure for the Intel 8080 microprocessor, including single-error correction capability, is, according to preliminary results, equal to about forty columns.

7.0 RESULTS AND CONCLUSIONS

A year (or so) ago we undertook the task of developing a set of basic tools for use in implementing built-in-test facilities in digital systems. It was stipulated that the BIT facilities should be capable of testing the system on-line continuously and without interfering with any of the normal, ongoing operations of the system. As a secondary objective, it was also considered very useful to arrive at BIT structures which one could implement without the need for special hardware; in other words, the BIT facilities should be a natural extension of the overall system design process utilizing the same modules used by the rest of the system. Another very important consideration, also, was the fact that all systems with BIT facilities should have the ability not only to detect but also to locate faults and then proceed to either mask or overcome such faults via an appropriate strategy of system reconfiguration.

The basic cell which we have proposed and the resulting cellular arrays satisfy all the aforementioned design objectives, and the notion of "hardware encoded" cellular arrays offers a new design tool in introducing routinely BIT facilities into a digital system. An additional advantage of the cellular array we have proposed is the ease with which one can design into a system reconfiguration capability either for the purpose of overcoming faults or for enhancing functional system flexibility. Finally, "hardware encoded" cellular structures and state-encoding schemes of the K-out-of-n type make possible the design of self-checkable checkers for on-line monitoring of the state-space and the output of cellular arrays.

Our research findings to-date point to several problems of immediate research interest. The order in which we present these problems in the sequel indicates the sequence in which, we feel, they should be undertaken if we are to demonstrate the utility and practical implications of our work in cellular arrays. We believe that such an objective at this point is justified, in view of the basic knowledge gained from our study of cellular structures, and we propose that reduction of our research findings to "physical realizations" be undertaken promptly in order to determine the manner in which, and also the extent to which, these findings have impacted the way we design digital modules or systems and their important characteristics such as reliability, testability,

maintainability, functional flexibility, speed, size, cost and production yields.

One way of evaluating the cellular structures, investigated in previous work, is by utilizing those structures as the basis of designing digital subsystems utilized in the implementation of well-understood computer architectures. As a starting point, we propose that an architecture similar to that employed in popular microprocessor chips be utilized to organize a microprocessor or a microcomputer system using cellular arrays of the type investigated earlier. The importance of this specific undertaking stems from the fact that it could be used as the basis for addressing a number of relevant questions through which we can firmly establish the utility of cellular structures in digital systems design.

The relevant questions of immediate interest to us may be stated as follows:

Question - 1 To what extent do design approaches based on cellular arrays influence overall system reliability, testability, and maintainability?

Here we propose to study the trade-offs between a real design configuration utilizing conventional design tools and its corresponding realization which is based on the programmable cellular structures which we have described in earlier reports [7]. System reliability will be determined parametrically in a way which accounts for the fact that in the proposed programmable cellular array one may allocate resources (i.e., columns of cells or just simply cells) for the purpose of realizing built-in-test (BIT) facilities in various different ways depending on the type of "hardware encoding" chosen, the way the cellular array is functionally segmented, and the budgetary constraints which are usually imposed on the designer of such a system. The budgetary constraints may be in terms of power requirements, physical size, speed, cost, etc. An additional parameter, of course, is the extent to which we wish to protect our system against the threats of a postulated fault environment (i.e., fault coverage). Encoding of the state-space and "random logic" portions of the system for error detection/correction will also be another very important consideration.

As one can easily deduce, the problem we are dealing with is an optimi-

zation problem involving a number of discrete parameters such as number of columns used for error detection/correction in the cellular structure, and the number of state variables required for encoding the state-space. A form of branch and bound technique will be used to obtain the solution to this problem. An alternate approach, of course, would be to approximate the discrete problem by a continuous one, proceed to estimate the error committed by such an approximation. The objective function in this optimization process may be either the mean-time to system failure or any measure of effectiveness that relates to system availability, mean-time to detect errors, throughput rates, etc.

System testability is, of course, another very important consideration, and the fact that a design cellular array is more amenable to systematic testing than subsystems utilizing random control logic should prove very useful. To what extent, however, uniformity in structure proves useful in the case of a system configured with cellular arrays of the type investigated in our previous work remains to be seen.

Finally, system maintainability will be addressed at two levels. First, through the use of built-in spares in the cellular array, and secondly, via system reconfiguration in the event of failure or perhaps a combination of the two. Graceful degradation in system performance will be central in all our considerations involving issues of system maintenance.

Question - 2 To what extent do design approaches based on cellular arrays influence system parameters such as speed, size and power consumption?

The answers to this question will determine the adequacy of the basic cell used and could very well result in design changes to the present cellular structure. The idea here is to develop a type of cell which is not only easily realizable, but one which has also the functional capability of serving in a competitive way as the basic building block for designing other more complex and well-understood computational structures.

System speed, size, and power consumption will also depend on how one organizes (i.e., maps) a machine structure on a cellular array. The mapping procedure which we developed for this purpose in previous work [7] will be used to make the required comparisons between conventionally designed systems and systems configured with cellular arrays.

Question - 3 To what extent do design approaches based on cellular arrays improve functional system flexibility?

Functional flexibility here refers to built-in system capability for purposes of reorganization in order to either overcome system failures or for the purpose of satisfying a changing computational environment. The extent to which this is practically feasible through the use of programmable cellular arrays needs to be investigated since the implications of these findings are potentially great from the standpoint of system maintainability, system reliability, and system availability. Furthermore, the possibility of putting the same hardware organization to multiple uses and the potential of having a system capable of degrading gracefully in the event of failure have a great deal of practical appeal from a designer's as well as the user's point of view.

The above three major questions deal with the problems which are of great practical significance to the designers and users of digital systems and should be the next topics of serious study.

REFERENCES

1. R. C. Minnick, "Cutpoint Cellular Logic," IEEE Trans. on Electronic Computers, vol. EC-13, pp. 685-698, Dec. 1964.
2. R. C. Minnick, "A Survey of Microcellular Research," J. Assoc. Comput. Mach., vol. 14, pp. 203-241, April 1967.
3. E. F. Codd, "Cellular Automata." New York and London: Academic Press, 1968.
4. W. H. Kautz, "Testing for Faults in Combinational Cellular Logic Arrays," Proc. 8th Ann. Symp. Switching and Automata Theory, Oct. 1967, pp. 161-174.
5. S. S. Yau and M. Orsic, "Fault Diagnosis and Repair of Cutpoint Cellular Arrays," IEEE Trans. on Computers, vol. C-19, No. 3, pp. 269-261.
6. F. B. Manning, "An Approach to Highly Integrated, Computer-Maintained Cellular Arrays," IEEE trans. on Computers, vol. C-26, No. 6, pp. 536-552, June, 1977.
7. E. W. Page and P. N. Marinos, "Programmable Array Realization of Synchronous Sequential Machines," IEEE Trans. on Computers, vol. C-26, No. 8, pp. 811-818, August 1977.
8. S. Mago, "Monotone Functions in Sequential Circuits," IEEE Trans. on Computers, vol. C-22, pp. 928-933, Oct., 1973.
9. T. Takaoka and T. Ibaraki, "N-Fail-Safe Sequential Machines," IEEE trans. on Computers vol. C-21, pp. 1189-1196, Nov. 1972.
10. Y. Tohma, Y. Ohyama, and R. Sakai, "Realization of Fail-Safe Sequential Machines by Using K-out-of-n Code," IEEE Trans. on Computers, vol. C-20, pp. 1270-1275, Nov. 1971.
11. E. W. Page, "Matrix Method for Systematizing Sequential-Circuit Design," Electronics Letters, pp. 49-50, vol. 12, No. 2, 22nd January 1976.

PRECEDING PAGE BLANK-NOT FILMED

APPENDIX

This appendix describes the salient points of the encoded version of the algorithm presented earlier in the report and it is used to carry out automatically the mapping of an arbitrary synchronous sequential machine into a cellular structure using the basic cell given in Figure 2.1.

The input format:

The first card is a description of the problem. It can be up to 80 characters long and can not be omitted.

Next, the machine parameters must be specified. Each parameter is in the form of an assignment statement that specifies both the value and the parameter to which it is to be assigned. Parameters are separated by a comma and/or one or more blanks. A semicolon must end the group. The order of the parameters is unimportant.

The parameters are:

- N - the number of states $N \geq 1$
- U - the number of input symbols $U \geq 1$
- W - the number of Mealy outputs $W \geq 0$
- V - the number of Moore outputs $V \geq 0$ $W + V \geq 1$
- K - the number of lines to a module $\binom{M}{K} \geq N$ $K \geq 0$
- M - the number of state variables $\binom{M}{K} \geq N$ $M \geq 1$
- VLENGTH - the maximum length of any figure

K, M, and VLENGTH are optional.

If K and/or M are not specified, a procedure is invoked to minimize M and/or K. (M is minimized first.)

The VLENGTH default should be sufficient for most jobs. If, however, a figure is printed out with the last columns missing, truncation has occurred and VLENGTH must be increased. The default is 250, increment by 1 for every column lost. If incrementing VLENGTH causes the IEN13011 OX NOT ENOUGH MAIN STORAGE AVAILABLE . . . message to occur, increase the Region parameter on the JOB card.

The next set of cards specify the machine state table. There should be $N(U(1+W)+V)$ inputs. The inputs are read in to fill a "standard" state table. That is, the present state numbers are from 1 to N (top to bottom), and the input symbols are from 1 to U (left to right). The next state can start

anywhere on the cards, followed immediately by a comma and immediately by the W Mealy lines, if $W > 0$. The Mealy outputs must be in the same order and contain no blanks between them. Follow this by one or more blanks and the next state, etc. This continues U times. The V Moore lines should follow the last set of W Moore lines, if $V > 0$, by exactly one space. The Moore outputs must be in the same order and contain no blanks between them. This procedure is repeated N times.

Last, the flip-flops are specified.

The possible types of flip-flops are:

- 1 - Delay
- 2 - Trigger
- 3 - SR
- 4 - JK

The specification of the flip-flops is of the form:

$HMNY_1 [,] TYPE_1 [[,] HMNY_2 [,] TYPE_2 . . .]$ where:

HMNY - the number of flip-flops of type TYPE $HMNY \geq 1$
(the iteration factor)

TYPE - the type of flip-flop $1 \leq TYPE \leq 4$

The total number of flip-flops specified, therefore, is $\sum HMNY$'s. The first $HMNY_1$ excitation functions are for flip-flops of type $TYPE_1$, the next $HMNY_2$ excitation functions are for flip-flops of type $TYPE_2$, and so forth. If the total number of flip-flops is greater than M, the excessive flip-flops are ignored. If the total number of flip-flops is less than M, the flip-flop specification list is reused. Suppose two delay flip-flops and one JK flip-flop are specified. If $M=2$, the machine will be realized entirely with delays. If $M=4$, the machine will be realized using three delays and one JK, the third excitation function will be for a JK flip-flop. (To use all of one type flip-flop, one could just specify $HMNY_1 = 1$ $TYPE_1 = \text{type.}$)

Limitations:

All parameters must be less than or equal to 32767, no matrix may be greater than 32767 characters in width, and there must be enough memory in the computer.

Example

P.S.	N.S., ZP		
	S1	S2	ZL
Q1	Q1,1	Q5,0	0
Q2	Q1,0	Q3,0	1
Q3	Q2,0	Q4,1	0
Q4	Q3,1	Q3,0	0
Q5	Q3,0	Q4,0	1

with delay flip-flops

input:

ILLUSTRATION OF ALGORITHM

N=5,U=2,W=1,V=1:

1,1 5,0 0 1,0 3,0 1 2,0 4,1 0
 3,1 3,0 0 3,0 4,0 1
 1 1

description of problem
 parameter specification
 state table
 specification
 flip-flop specification

SOURCE LISTING

STMT LEV HT

```

1      0  PAFOSSM: PROCEDURE OPTIONS (MAIN);
      /* THIS IS A PROCEDURE FOR AUTOMATICALLY MAPPING AN ARBITRARY
      SYNCHRONOUS SEQUENTIAL MACHINE INTO A CELLULAR ARRAY */
      /* VLENGTH IS THE ABSOLUTE MAXIMUM NUMBER OF CHARACTERS THAT CAN
      BE INCLUDED IN ANY ROW OF AN OUTPUT FIGURE */
2      1  0  DCL VLENGTH FIXED BINARY INITIAL (250);
      /* LEVEL IS THE MAXIMUM NUMBER OF CHARACTERS TO BE INCLUDED
      IN EACH PRINTED LINE */
3      1  0  DCL LEVEL FIXED BINARY INITIAL (120);
4      1  0  DCL (ZPX,ZPY,ZLX,ZLY) FIXED BINARY;
5      1  0  DCL (N,U,M,W,V,K) FIXED BINARY;
      /* PRINT DESCRIPTION OF PROBLEM */
6      1  0  BEGIN;
7      2  0      DCL LINE CHARACTER (80);
8      2  0      GET EDIT (LINE) (A(80));
9      2  0      PUT PAGE EDIT(LINE) (A(80));
10     2  0  END;
      /* GET MACHINE PARAMETERS AND PRINT */
11     1  0  N,U,M = 0; W,V,K = -1;
13     1  0  GET DATA (N,U,M,W,V,K,M,VLENGTH);
14     1  0  CALL CALCPARAM;
15     1  0  PUT SKIP(4) LIST('MACHINE PARAMETERS:');
16     1  0  PUT SKIP EDIT('||NUMCHAR(N)|| STATES, ||NUMCHAR(U)||
      ' INPUT SYMBOLS, ||NUMCHAR(W)|| MEALY OUTPUTS, ||
      NUMCHAR(V)|| MOORE OUTPUTS, AND ||NUMCHAR(K)||
      '-OUT-OF-||NUMCHAR(M)|| ENCODING USED.') (1);
      /* MAIN CONTROL SECTION */
17     1  0  BEGIN;
18     2  0      DCL 1 FFPDDE BASED (FFP),
      2 INFO,
      3 FFPMMY FIXED BINARY,
      3 FFPYFE FIXED BINARY,
      2 LINK POINTER,
      (FFPD,HEAD) POINTER;
19     2  0      DCL NEXTST(N,U) FIXED BINARY;
20     2  0      DCL ZP(ZPY,ZPX) BIT(1);
21     2  0      DCL ZL(ZLX,ZLY) BIT(1);
22     2  0      DCL STAB(N+3) CHARACTER (VLENGTH) VARYING;
23     2  0      DCL WCHST(N,W) BIT(1);
24     2  0      DCL (MAYLN,STAB,STARTS,WA,ELINE,EDENT) FIXED BINARY;
25     2  0      CALL READIN; /* READ IN STATE TABLE AND FLIP FLOP LIST */
26     2  0      CALL PFFF; /* PRINT FLIP FLOP LIST */
27     2  0      CALL PESTTAB; /* PRINT STATE TABLE */
28     2  0      CALL PESTASG; /* PRINT STATE ASSIGNMENTS */
29     2  0      CALL PPDJ; /* PRINT PARTITIONS PDJ */
30     2  0      CALL PCONNMAT; /* PRINT CONNECTION MATRIX */
31     2  0      CALL POUTNMAT; /* PRINT OUTPUT MATRIX */
32     2  0      CALL PEMMMAT; /* PRINT MEMORY ELEMENT MATRIX */
33     2  0      CALL PPSPECMAT; /* PRINT CELL SPECIFICATION-FUNCTION MATRIX */
34     2  0      CALL PPERNMAT; /* PRINT EXCITATION AND INHIBIT MATRICES */
      /* READ IN STATE TABLE AND FLIP FLOPS, GET HEAD

```

PARCOURS : 3300000000 00000000 (3300000000) :

```

OPTION NEXT STATE, ITERATION FACTOR, FLOW FLOW TYPE, COMPLETE
SPECIFICATION = /
35 2 0 | REAPIN: PROCEDURE:
/* GLOBAL: N, J, NEXTST, ZP, ZI, W, V, HEAD, FFP, FEMODE */
36 3 0 | DCL (I, J, L, NST, HMMY, TYPE) FIXED BINARY;
37 3 0 | DCL TAIL POINTER;
38 3 0 | DCL NULL BUILDED;
39 3 0 | ON ENDFILE BEGIN;
40 4 0 | PUT SKIP LIST('***ERROR*** STATE TABLE INCOMPLETELY SPECIFIED'
);
41 4 0 | STOP;
42 4 0 | END;
43 3 0 | DO I = 1 TO N;
44 3 1 | DO J = 1 TO 3;
/* GET NEXT STATE, N >= NEXT STATE >= 1 */
45 3 2 | GET LIST(NST);
46 3 2 | IF NST <= 0 | NST > N THEN
DO;
47 3 3 | PUT SKIP LIST('***ERROR*** NEXT STATE OUT OF RANGE, =',
, NST);
48 3 3 | STOP;
49 3 3 | END;
50 3 2 | NEXTST(I, J) = NST;
/* GET MOORE OUTPUTS, IF ANY */
51 3 2 | IF W > 0 THEN
GET EDIT((ZP(I, L) DO L = (J-1)*W+1 TO J*W)) ((W)B(1));
52 3 2 | END;
/* GET MOORE OUTPUTS, IF ANY */
53 3 1 | IF V > 0 THEN
GET EDIT(ZI(I, *)) (X(1), (V)B(1));
54 3 1 | END;
55 3 0 | HEAD = NULL;
56 3 0 | ON ENDFILE BEGIN;
57 4 0 | PUT SKIP LIST('***ERROR*** NO FLOW FLOPS SPECIFIED');
58 4 0 | STOP;
59 4 0 | END;
/* GET ITERATION FACTOR, HMMY MUST BE > 0 */
60 3 0 | GET LIST(HMMY);
61 3 0 | IF HMMY <= 0 THEN
DO;
62 3 1 | PUT SKIP LIST('***ERROR*** FLOW FLOW ITERATION FACTOR OUT OF
RANGE, =', HMMY) (2 A, F(10));
63 3 1 | STOP;
64 3 1 | END;
65 3 0 | ON ENDFILE BEGIN;
66 4 0 | PUT SKIP LIST('***ERROR*** FLOW FLOW TYPE NOT SPECIFIED');
67 4 0 | STOP;
68 4 0 | END;
/* GET FLOW FLOW TYPE, 4 >= TYPE >= 1 */
69 3 0 | GET LIST(TYPE);
70 3 0 | IF TYPE <= 0 | TYPE > 4 THEN
DO;
71 3 1 | PUT SKIP LIST('***ERROR*** FLOW FLOW TYPE NOT SPECIFIED,
TYPE=', TYPE) (2 A, F(10));

```

STMT LEV WT

```

72 3 1 | STOP;
73 3 1 | END;
/* GET NODE, LINK AND FILL IN INFORMATION */
74 3 0 | ALLOCATE FFNODE;
75 3 0 | IF HEAD = NULL THEN
76 3 1 | DO;
77 3 1 | HEAD = FFP; TAIL = FFP;
78 3 1 | END;
79 3 0 | ELSE
80 3 1 | DO;
81 3 1 | TAIL->FFNODE.LINK = FFP; TAIL = FFP;
82 3 1 | END;
83 3 0 | FFMNY = HMNY; FFTYPE = TYPE;
84 3 0 | ON ENDFILE GOTO ENDF;
85 3 0 | GOTO GETFF;
/* MAKE INTO CIRCULARLY LINKED LIST */
86 3 0 | ENDF: TAIL->FFNODE.LINK = HEAD;
87 3 0 | END READING;
/* PRINT FLIP FLOP LIST */
88 2 0 | PFFT: PROCEDURE;
89 3 0 | /* GLOBAL: HEAD, BLINK, FFNODE */
90 3 0 | DCL FNAME(4) CHARACTER(2) VARYING INITIAL('D','I','S','K');
91 3 0 | DCL (HMNY,TYPE) FIXED BINARY;
92 3 0 | DCL TP POINTER;
93 3 0 | PUT SKIP(3) LIST('FLIP FLOP LIST:');
94 3 0 | TP = HEAD;
95 3 0 | PUT SKIP;
96 3 0 | /* INITIALIZE PRINTLINE */
97 3 0 | BLINK = 0;
98 3 0 | CALL PRINTLINE(' ',0);
/* LIST ALL FLIP FLOPS AND ITERATION FACTORS */
99 3 0 | IF: HMNY = TP->FFNODE.FFMNY; TYPE = TP->FFNODE.FFTYPE;
100 3 0 | CALL PRINTLINE(CHARACTER(HMNY) || ' ' || FNAME(TYPE) || ' ' || ' ');
101 3 0 | TP = TP->FFNODE.LINK;
102 3 0 | IF TP = HEAD THEN
103 3 0 | RETURN;
104 3 0 | GOTO IF;
105 3 0 | END PFFT;
/* PRINT TREE FORMAT LINE, TAKING CARE OF LINE OVERFLOW */
106 2 0 | PRINTLINE: PROCEDURE(LINE,ENCL);
107 3 0 | /* GLOBAL: BLINK,IDENT,LEVEL */
108 3 0 | DCL LIND CHARACTER(*) VARYING,
109 3 0 | ENCL FIXED BINARY;
110 3 0 | /* ENCL IS HOW MANY CHARACTERS TO INCREASE ARE REMAINING
111 3 0 | IN OUTPUT LINE */
112 3 0 | DCL (LL,SL,ALINE,CL) FIXED BINARY;
113 3 0 | CL = 1;
114 3 0 | LL = LENGTH(LINE);
115 3 0 | /* IF FIRST CALL FOR THIS INVOCATION - SAVE INDENTATION LENGTH */
116 3 0 | IF BLINK = 0 THEN
117 3 0 | IDENT = LL;
118 3 0 | ALINE = LIND-IDENT-ENCL;
119 3 0 | /* LIND > LEVEL */
120 3 0 | DO WHILE (LL > ALINE);

```

STATE LEV NT

```

113 3 1 | CL = LEVEL-BLINE;
114 3 1 | PUT EDIT(SUBSTR(LINE,SL,CL)) (A);
115 3 1 | BLINE = IDENT;
116 3 1 | PUT SKIP EDIT(REPEAT(' ',BLINE-1)) (A);
117 3 1 | LL = LL-CL;
118 3 1 | SL = SL+CL;
119 3 1 | END;
/* LINE > REMAINING OUTPUT LINE */
120 3 0 | IF BLINE+LL > LEVEL-ENCL THEN
DO;
121 3 1 | BLINE = LL+IDENT;
122 3 1 | PUT SKIP EDIT(REPEAT(' ',IDENT-1)) (A);
123 3 1 | END;
124 3 0 | ELSE
BLINE = LL+BLINE;
125 3 0 | PUT EDIT(SUBSTR(LINE,SL,LL)) (A);
126 3 0 | END PRINTLINE;
/* PRINT STATE TABLE AND SET STARTS,NA */
127 2 0 | PRESTAB: PROCEDURE;
/* GLOBAL: OTAB,N,MAXLN,START,STARTS,NA,U,W,V,NEXTST,ED,ZL */
128 3 0 | DCL (I,J,H,II) FIXED BINARY;
/* INITIALIZE PRESENT STATE COLUMNS */
129 3 0 | OTAB(1) = '1';
130 3 0 | DO I = 3 TO N+2;
131 3 1 | OTAB(I) = '0' || NUMCHAR(I-2) || '1';
132 3 1 | END;
133 3 0 | MAXLN = LENGTH(OTAB(N+2));
134 3 0 | DO I = 1,3 TO N+2;
135 3 1 | J = MAXLN-LENGTH(OTAB(I))-1;
136 3 1 | IF J < 0 THEN
GOTO DONE;
137 3 1 | OTAB(I) = REPEAT(' ',J) || OTAB(I);
138 3 1 | END;
139 3 0 | DONE: START = MAXLN;
140 3 0 | STARTS = START;
141 3 0 | NA = CEIL(MAXLN/2)-1;
142 3 0 | REPEAT(OTAB(1),NA+1,1) = 'V';
/* FILL IN NEXT STATE AND WEALY OUTPUT (IF ANY) */
143 3 0 | DO J = 1 TO U;
144 3 1 | OTAB(1) = OTAB(1) || 'S' || NUMCHAR(J);
145 3 1 | DO I = 1 TO N;
146 3 2 | II = I+2;
147 3 2 | OTAB(II) = OTAB(II) || 'Q' || NUMCHAR(NEXTST(I,J));
148 3 2 | IF W > 0 THEN
DO;
149 3 3 | OTAB(II) = OTAB(II) || ' ';
150 3 3 | DO H = (J-1)*W+1 TO J*W;
151 3 4 | OTAB(II) = OTAB(II) || REP(I,H);
152 3 4 | END;
153 3 2 | END;
154 3 2 | END;
/* RIGHT JUSTIFY NEXT STATE (OP) */
155 3 1 | CALL GETMAXLN;
156 3 1 | CALL BRADJUST(*);

```

STATE DEV NT

```

157 3 1 | END;
158 3 0 | /* WILL IN MOORE OUTPUTS, IF ANY */
159 3 0 | IF V > 0 THEN
160 3 1 | DO;
161 3 1 |   OTAB(1) = OTAB(1) || ' ' || REPEAT(' ',V);
162 3 1 |   DO I = 1 TO N;
163 3 2 |     II = I+2;
164 3 2 |     OTAB(II) = OTAB(II) || ' ' || ' ';
165 3 2 |     DO H = 1 TO V;
166 3 3 |       OTAB(II) = OTAB(II) || ZL(I,H);
167 3 3 |     END;
168 3 2 |   END;
169 3 1 | END;
170 3 0 | /* OUTPUT TABLE AFTER INSERTING BOUNDARIES */
171 3 0 | J = LENGTH(OTAB(1))-1;
172 3 0 | OTAB(2) = REPEAT(' ',J);
173 3 0 | OTAB(N+3) = REPEAT(' ',J);
174 3 0 | PUT PAGE LIST('STATE TABLE');
175 3 0 | PUT SKIP(4) EDIT('PRESENT', 'STATE' NEXT STATE) (1,OTAB,A);
176 3 0 | IF W > 0 THEN
177 3 0 |   PUT EDIT(' ',ZP) (A);
178 3 0 | IF V > 0 THEN
179 3 0 |   PUT EDIT(' ',ZL) (A);
180 3 0 | PUT SKIP EDIT(' ') (X(NA),A);
181 3 0 | CALL PRINTTAB;
182 3 0 | END PRESTAB;
183 2 0 | /* FIND PRESENT MAXIMUM LENGTH OF OTAB AND SET MAXLN */
184 2 0 | GETMAXLN: PROCEDURE;
185 3 0 | /* GLOBAL: MAXLN,OTAB,N */
186 3 0 | DCL I FIXED BINARY;
187 3 0 | MAXLN = LENGTH(OTAB(1));
188 3 0 | DO I = 3 TO N+2;
189 3 1 |   IF LENGTH(OTAB(I)) > MAXLN THEN
190 3 1 |     MAXLN = LENGTH(OTAB(I));
191 3 1 | END;
192 3 0 | END GETMAXLN;
193 3 0 | /* ADJUST COLUMN STARTING AT START+1, ENDING AT MAXLN, UPDATE START
194 3 0 | IF CF = 1 RIGHT JUSTIFY, IF CF = 2 CENTER JUSTIFY */
195 2 0 | PREADJUST: PROCEDURE(CF);
196 3 0 | /* GLOBAL: N,OTAB,MAXLN,START */
197 3 0 | DCL (I,J,L,II,CF) FIXED BINARY;
198 3 0 | DO I = 1,3 TO N+2;
199 3 1 |   L = MAXLN-LENGTH(OTAB(I));
200 3 1 |   J = TRIL(L/CF);
201 3 1 |   II = L-J-1;
202 3 1 |   OTAB(I) = SUBSTR(OTAB(I),1,START) || REPEAT(' ',J) ||
203 3 1 |     SUBSTR(OTAB(I),START+1);
204 3 1 |   IF II >= 0 THEN
205 3 1 |     OTAB(I) = OTAB(I) || REPEAT(' ',II);
206 3 1 | END;
207 3 0 | START = MAXLN+1;
208 3 0 | END PREADJUST;
209 2 0 | /* PRINT OTAB, TAKE CARE OF OVERFLOW */
210 2 0 | PRINTTAB: PROCEDURE;

```


SCM: LBN NT

```

197 3 0 | /* GLOBAL: OTAB,LEVEL,N */
198 3 0 | DCL (J,L,I,IL) FIXED BINARY;
199 3 0 | SL = 1;
200 3 0 | J = LENGTH(OTAB(1));
201 3 0 | NP: IF J > LEVEL THEN
202 3 0 |     L = LEVEL;
203 3 0 | ELSE
204 3 0 |     L = J;
205 3 0 | J = J-LEVEL;
206 3 0 | DO I = 1 TO N+3;
207 3 1 |     PUT SKIP PDIT(SUBSTR(OTAB(I),SL,L)) (A);
208 3 1 | END;
209 3 0 | IF J <= 0 THEN
210 3 0 |     RETURN;
211 3 0 | PUT PAGE LIST(' ');
212 3 0 | PUT SKIP(6);
213 3 0 | SL = SL+L;
214 3 0 | GOTO NP;
215 3 0 | END PRINTMAC;
216 3 0 | /* PRINT STATE ASSIGNMENT */
217 2 0 | PFS: PROCEDURE;
218 3 0 | /* GLOBAL: N,STARTS,OTAB,START,X,K,MACHST,N */
219 3 0 | DCL (I,J) FIXED BINARY;
220 3 0 | /* RESET OTAB TO PRESENT STATE COLUMN */
221 3 0 | DO I = 1,3 TO N+2;
222 3 1 |     OTAB(I) = SUBSTR(OTAB(I),1,STARTS);
223 3 1 | END;
224 3 0 | START = STARTS;
225 3 0 | /* GENERATE STATE ASSIGNMENTS */
226 3 0 | CALL GETMACHST;
227 3 0 | /* FILL STATE ASSIGNMENTS INTO TABLE */
228 3 0 | DO J = 1 TO N;
229 3 1 |     OTAB(1) = OTAB(1) || 'Y' || NUMCHAR(J);
230 3 1 |     DO I = 3 TO N+2;
231 3 2 |         OTAB(I) = OTAB(I) || MACHST(I-2,J);
232 3 2 |     END;
233 3 1 |     CALL GETMAXIM;
234 3 1 |     CALL READJUST(2);
235 3 1 | END;
236 3 0 | /* OUTPUT TABLE AFTER INSERTING BOUNDARIES */
237 3 0 | J = LENGTH(OTAB(1))-1;
238 3 0 | OTAB(2) = REPEAT('-',J);
239 3 0 | OTAB(N+3) = REPEAT(' ',J);
240 3 0 | PUT PAGE LIST
241 3 0 |     (NUMCHAR(N) || '-PUT-OF-' || NUMCHAR(N) || ' STATE ASSIGNMENTS:');
242 3 0 | PUT SKIP(4) EDIT('PRESENT', 'STATE STATE VARIABLE', ' ');
243 3 0 |     (A,SKIP,3,SKIP,7(NA),3);
244 3 0 | CALL PRINTMAC;
245 3 0 | END PFS:;
246 2 0 | /* GENERATE N WAYS OF X CHOOSE K */
247 2 0 | GETMACHST: PROCEDURE;
248 3 0 | /* GLOBAL: MACHST,N,X */
249 3 0 | DCL (I,J,L,M) FIXED BINARY;
250 3 0 | MACHST(1,*) = '000';

```

PL/1 OPTIMIZING COMPILER

PAROBSY: PROCEDURE OPTIONS (MAIN);

SYN: LEV MT

```

237 3 0 | DO I = 1 TO N;
238 3 1 |   MACHST(1,I) = '1'B;
239 3 1 | END;
240 3 0 | DO I = 2 TO N;
241 3 1 |   J = M;
242 3 1 |   DO WHILE (~MACHST(I-1,J));
243 3 2 |     MACHST(I,J) = '0'B;
244 3 2 |     J = J-1;
245 3 2 |   END;
246 3 1 |   IF J = M THEN
247 3 2 |     DO:
248 3 2 |       MACHST(I,J+1) = '1'B; MACHST(I,J) = '0'B;
249 3 2 |     END;
250 3 1 |   ELSE
251 3 2 |     DO:
252 3 2 |       J = M-1;
253 3 2 |       L = 1;
254 3 2 |       DO WHILE (MACHST(I-1,J));
255 3 3 |         L = L+1;
256 3 3 |         J = J-1;
257 3 3 |       END;
258 3 2 |       DO WHILE (~MACHST(I-1,J));
259 3 3 |         J = J-1;
260 3 3 |       END;
261 3 2 |       MACHST(I,J) = '0'B;
262 3 2 |       DO IM = J+1 TO J+1+L;
263 3 3 |         MACHST(I,IM) = '1'B;
264 3 3 |       END;
265 3 2 |       DO IM = J+2+1 TO M;
266 3 3 |         MACHST(I,IM) = '0'B;
267 3 3 |       END;
268 3 2 |     END;
269 3 1 |   DO IM = 1 TO J-1;
270 3 2 |     MACHST(I,IM) = MACHST(I-1,IM);
271 3 2 |   END;
272 3 1 | END GETMACHST;
273 2 0 | /* PRINT PARTITIONS PCU */
274 3 1 | PARTO: PROCEDURE;
275 3 0 |   /* GLOBAL: N,BLINT,N,MACHST */
276 3 0 |   DO (N,M) FIXED BINARY;
277 3 0 |   DO (FLAG,FIRST) FILL(1);
278 3 0 |   PUT FILL LIST('PARTITIONS PCU=(PCU:PCU)');
279 3 0 |   PUT SKIP(2);
280 3 1 |   DO J = 1 TO N;
281 3 1 |     PUT SKIP(2);
282 3 1 |     /* INITIALIZER PRINTLINE */
283 3 1 |     BLINE = 0;
284 3 1 |     CALL PRINTLINE('PCU INUMCHAP(J) I I'=(J,0);
285 3 1 |     /* FLAG = 0 FOR PCU, FLAG = 1 FOR BLD */
286 3 1 |     FLAG = '0'B;
287 3 1 |     FIRST = '1'B;
288 3 1 |     DO I = 1 TO N;
289 3 2 |       IF MACHST(I,J) = FLAG THEN

```

STMT LEV WT

```

286 3 3 |         IF FIRST THEN
287 3 3 |             DO;
288 3 3 |                 CALL PRINTLINE(NUMLCHAR(I),1);
289 3 2 |                 FIRST = '1'B;
290 3 3 |             END;
291 3 3 |         ELSE
292 3 3 |             DO;
293 3 2 |                 CALL PRINTLINE(' ',0);
294 3 1 |                 CALL PRINTLINE(NUMLCHAR(I),1);
295 3 1 |             END;
296 3 2 |         IF FLAG THEN
297 3 2 |             CALL PRINTLINE(' ',0);
298 3 2 |         ELSE
299 3 2 |             DO;
300 3 1 |                 FLAG = '1'B;
301 3 0 |                 CALL PRINTLINE(' ',0);
302 2 0 |                 GOTO LP;
303 3 0 |             END;
304 3 0 |         END;
305 3 0 |     END BEPIJ;
306 3 0 | /* PRINT CONNECTION MATRIX */
307 3 0 | PROCNMAT: PROCEDURE;
308 3 0 |     /* GLOBAL: N,OTAB,STARTS,STAB,NA */
309 3 0 |     DO J = 1 TO N;
310 3 1 |         /* RESET OTAB TO PRESENT STATE COLUMN */
311 3 1 |         DO J = 1,3 TO N*2;
312 3 1 |             OTAB(J) = SUBSTR(OTAB(J),1,STARTS);
313 3 1 |         END;
314 3 0 |         STAB = STARTS;
315 3 0 |         /* FORM CONNECTION MATRIX */
316 3 0 |         DO J = 1 TO N;
317 3 1 |             OTAB(1) = OTAB(1) || 'Q' || NUMCHAR(J);
318 3 1 |             CALL GETSCOL(OTAB,J);
319 3 1 |         END;
320 3 0 |         /* OUTPUT MATRIX AFTER INSERTING BOUNDARIES */
321 3 0 |         OTAB(1) = OTAB(1) || ' ' || ' ';
322 3 0 |         DO J = 2 TO N*2;
323 3 1 |             OTAB(J) = OTAB(J) || ' ' || ' ';
324 3 0 |         END;
325 3 0 |         OTAB(2) = REPEAT(' ',STARTS-2) || ' ' || REPEAT(' ',LENGTH(OTAB(1))
326 3 0 |             -2-STARTS) || ' ' || ' ';
327 3 0 |         OTAB(N*3) = OTAB(2);
328 2 0 |         PUT PAGE LIST('CONNECTION MATRIX');
329 3 0 |         PUT SKIP(4) EDIT('PRESENT',STATE, 'NEXT STATE',1)
330 3 0 |             (A,SKIP,1,SKIP,X(NA),1);
331 3 0 |         CALL PRINTMAT;
332 3 0 |     /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR CONNECTION MATRIX */
333 3 0 |     COMF: PROCEDURE(H,I,J) RETURNS (R1)
334 3 0 |     /* GLOBAL: NEXTST */
335 3 0 |     DO (H,I,J) FIXED BINARY;
336 3 0 |     RETURN(NEXTST(2,H) = J);
337 3 0 |     END COMF;
338 3 0 |     END PROCNMAT;

```

STATE LEV HT

```

/* FORM COLUMN WITH PROPER INPUT SYMBOLS - FUNC TELLS WHICH
   INPUT SYMBOLS TO USE */
326 2 0 GETSCOL: PROCEDURE(FUNC,J):
      /* GLOBAL: N,U,OTAB */
327 3 0   DCL FUNC ENTRY RETURNS (BIT(1));
328 3 0   DCL (I,II,H,J) FIXED BINARY;
329 3 0   DCL FIRST BIT(1);
330 3 0   DO I = 1 TO N;
331 3 1     II = I+2;
      /* FORM PARTICULAR MATRIX ENTRY */
332 3 1     FIRST = '1'B;
333 3 1     DO H = 1 TO U;
334 3 2       IF FUNC(H,I,J) THEN
         IF FIRST THEN
335 3 3           OTAB(II) = OTAB(II) || 'S' || NUMCHAR(H);
336 3 3           FIRST = '0'B;
337 3 3         END;
338 3 2       ELSE
         OTAB(II) = OTAB(II) || 'S' || NUMCHAR(H);
339 3 2     END;
340 3 1     IF FIRST THEN
         OTAB(II) = OTAB(II) || '0';
341 3 1     END;
      /* CENTER JUSTIFY COLUMN */
342 3 0     CALL GETMAXLN;
343 3 0     CALL READJUST(2);
344 3 0     END GETSCOL;
/* PRINT OUTPUT MATRIX */
345 2 0 PRINTMAT: PROCEDURE;
      /* GLOBAL: N,OTAB,STARTS,STAFF,NA */
346 3 0   DCL J FIXED BINARY;
      /* RESET OTAB TO PRESENT STATE COLUMN */
347 3 0   DO J = 1,3 TO N+2;
348 3 1     OTAB(J) = SUBSTR(OTAB(J),*,STARTS);
349 3 1   END;
350 3 0   STAFF = STARTS;
      /* FORM OUTPUT MATRIX */
351 3 0   CALL OUTMAT;
      /* OUTPUT MATRIX AFTER INSERTING BOUNDARIES */
352 2 0   PUT PAGE LIST('OUTPUT MATRIX');
353 3 0   PUT SKIP(4) EDIT('PRESENT',STATE,PRIMARY CURRENT,'||'
      (1,SKIP,1,SKIP,X(NA),2);
354 3 0   CALL PRINTMAT;
355 3 0   END PRINTMAT;
/* FORM OUTPUT MATRIX */
356 2 0 OUTMAT: PROCEDURE;
      /* GLOBAL: N,OTAB,V,N,ZI,STARTS */
357 3 0   DCL (J,I) FIXED BINARY;
      /* FORM NEARLY OUTPUT FUNCTIONS */
358 3 0   DO J = 1 TO N;
359 3 1     OTAB(1) = OTAB(1) || 'P' || NUMCHAR(J);
360 3 1     CALL GETSCOL(OUTF,J);
361 3 1   END;

```

STMT LBY MT

```

362 3 0 /* FORM MODES OUTPUT FUNCTIONS */
363 3 1 DO J = 1 TO V;
364 3 1 OTAB(1) = OTAB(1) || ZL || NUMCHAR(J);
365 3 2 DO I = 3 TO N+2;
366 3 2 OTAB(I) = OTAB(I) || ZI(I-2,J);
367 3 1 END;
368 3 1 /* CENTER JUSTIFY COLUMN */
369 3 1 CALL GETMAXLN;
370 3 0 OTAB(1) = OTAB(1) || ' ';
371 3 0 DO J = 3 TO N+2;
372 3 1 OTAB(J) = OTAB(J) || ' ';
373 3 1 END;
374 3 0 OTAB(2) = REPEAT(' ',STARTS-2) || '-' || REPEAT(' ',LENGTH(OTAB(1))
375 3 0 -2-STARTS) || '-';
376 3 0 OTAB(N+3) = OTAB(2);
377 3 0 /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR HEAVY MATRIX */
378 3 0 OUTP: PROCEDURE(H,I,J) RETURNS (BIT(1));
379 3 0 /* GLOBAL: ZP,W */
380 3 0 DCL (H,I,J) FIXED BINARY;
381 3 0 RETURN(ZP(C,(H-1)*W+J));
382 3 0 END OUTP;
383 3 0 END OUTMAT;
384 3 0 /* PRINT MEMORY ELEMENT MATRIX */
385 3 0 PPMEMMAT: PROCEDURE;
386 3 0 /* GLOBAL: M,OTAB,STARTS,STATE,FFNODE,HEAD,M,NA */
387 3 0 DCL (HMMV,TYPE,J) FIXED BINARY;
388 3 0 /* RESET OTAB TO PRESENT STATE COLUMN */
389 3 0 DO J = 1,3 TO N+2;
390 3 1 OTAB(J) = SUBSTR(OTAB(J),'',STARTS);
391 3 1 END;
392 3 0 STATE = STARTS;
393 3 0 /* USE HMMV OF TYPE FLIP FLOPS UNTIL * USED */
394 3 0 J = 1;
395 3 0 HMMV = HEAD->FFNODE.FFHMMV; TYPE = HEAD->FFNODE.FFTYPE;
396 3 0 INFF: IF TYPE = 1 THEN
397 3 1 CALL DELAY;
398 3 1 ELSE
399 3 1 IF TYPE = 2 THEN
400 3 2 CALL TRIGGER;
401 3 1 ELSE
402 3 1 IF TYPE = 3 THEN
403 3 2 CALL SF;
404 3 1 ELSE
405 3 1 CALL JK;
406 3 0 IF J = M THEN
407 3 1 GOTO OFF;
408 3 0 J = J+1;
409 3 0 IF HMMV = * THEN
410 3 1 DO;
411 3 2 HEAD = HEAD->FFNODE.LINK;
412 3 2 HMMV = HEAD->FFNODE.FFHMMV; TYPE = HEAD->FFNODE.FFTYPE;

```

SYMT LET MT

```

400 3 1 |      END;
401 3 0 |      ELSE
402 3 0 |      HMMY = HMMY-1;
403 3 0 |      GOTO MFF;
404 3 0 |      /* OUTPUT MATRIX AFTER INSERTING BOUNDARIES */
405 3 0 |      DFF: OTAB(1) = OTAB(1) || ' ';
406 3 0 |      DO J = 3 TO N+2;
407 3 1 |      OTAB(J) = OTAB(J) || ' ';
408 3 1 |      END;
409 3 0 |      OTAB(2) = REPEAT(' ', STARTS-2) || '-' || REPEAT(' ', LENGTH(OTAB(1))
410 3 0 |      -2-STARTS) || '-';
411 3 0 |      OTAB(N+3) = OTAB(2);
412 3 0 |      PUT PAGE LIST (MEMORY ELEMENT MATRIX);
413 3 0 |      PUT SKIP(4) FORT('PRESENT', STATE MEMORY ELEMENT, ' ');
414 3 0 |      (X, SKIP, 1, SKIP, Y(N), 1);
415 3 0 |      CALL PRINTMA;
416 3 0 |      /* FORM DELAY EXCITATION COLUMN */
417 3 0 |      DELAY: PROCEDURE;
418 3 0 |      /* GLOBAL: OTAB, J */
419 3 0 |      OTAB(1) = OTAB(1) || 'D' || NUMCHAR(J);
420 3 0 |      CALL GETSCOL(DELAY, J);
421 3 0 |      /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR DELAY FLIP FLOP */
422 3 0 |      DELAY: PROCEDURE(H, I, J) RETURNS (BIT(1));
423 3 0 |      /* GLOBAL: MACHST, NEXTST */
424 3 0 |      DCL (H, I, J) FIXED BINARY;
425 3 0 |      RETURN(MACHST(NEXTST(2, H), J));
426 3 0 |      END DELAY;
427 3 0 |      /* FORM TRIGGER EXCITATION COLUMN */
428 3 0 |      TRIGGER: PROCEDURE;
429 3 0 |      /* GLOBAL: OTAB, J */
430 3 0 |      OTAB(1) = OTAB(1) || 'T' || NUMCHAR(J);
431 3 0 |      CALL GETSCOL(TRIGGER, J);
432 3 0 |      /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR TRIGGER FLIP FLOP */
433 3 0 |      TRIGGER: PROCEDURE(H, I, J) RETURNS (BIT(1));
434 3 0 |      /* GLOBAL: MACHST, NEXTST */
435 3 0 |      DCL (H, I, J) FIXED BINARY;
436 3 0 |      RETURN(MACHST(NEXTST(2, H), J) - MACHST(1, J));
437 3 0 |      END TRIGGER;
438 3 0 |      /* FORM SF EXCITATION COLUMNS */
439 3 0 |      SF: PROCEDURE;
440 3 0 |      /* GLOBAL: OTAB, J */
441 3 0 |      OTAB(1) = OTAB(1) || 'S' || NUMCHAR(J);
442 3 0 |      CALL GETSCOL(SF, J);
443 3 0 |      OTAB(1) = OTAB(1) || 'E' || NUMCHAR(J);
444 3 0 |      CALL GETSCOL(SF, J);
445 3 0 |      /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR S(F) FLIP FLOP */
446 3 0 |      SF: PROCEDURE(H, I, J) RETURNS (BIT(1));
447 3 0 |      /* GLOBAL: MACHST, NEXTST */
448 3 0 |      DCL (H, I, J) FIXED BINARY;
449 3 0 |      RETURN(MACHST(1, J) & MACHST(NEXTST(2, H), J));
450 3 0 |      END SF;
451 3 0 |      /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR S(F) FLIP FLOP */

```

PROCESSING: PROCEDURE OPTIONS (MAY) :

```

437 4 0 | IFF: PROCEDURE(H,I,J) RETURNS (BIT(1));
438 4 0 | /* GLOBAL: MACHST,NEXTST */
439 4 0 | DCL (H,I,J) FIXED BINARY;
440 4 0 | RETURN(MACHST(I,J) & ~MACHST(NEXTST(I,H),J));
441 4 0 | END IFF;
442 3 0 | JK: PROCEDURE:
443 4 0 | /* GLOBAL: OTAB,J */
444 4 0 | OTAB(1) = OTAB(1) || 'J' || INTNCHAR(J);
445 4 0 | CALL GETSCOL(JF,J);
446 4 0 | OTAB(1) = OTAB(1) || 'K' || INTNCHAR(J);
447 4 0 | CALL GETSCOL(KF,J);
448 4 0 | /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR J(K) FLIP FLOP */
449 4 0 | JF: PROCEDURE(H,I,J) RETURNS (BIT(1));
450 4 0 | /* GLOBAL: MACHST,NEXTST */
451 4 0 | DCL (H,I,J) FIXED BINARY;
452 4 0 | RETURN(~MACHST(I,J) & MACHST(NEXTST(I,H),J));
453 4 0 | END JF;
454 4 0 | /* FUNCTION TO CHOOSE INPUT SYMBOLS FOR K(N) FLIP FLOP */
455 4 0 | KF: PROCEDURE(H,I,J) RETURNS (BIT(1));
456 4 0 | /* GLOBAL: MACHST,NEXTST */
457 4 0 | DCL (H,I,J) FIXED BINARY;
458 4 0 | RETURN(MACHST(I,J) & ~MACHST(NEXTST(I,H),J));
459 4 0 | END KF;
460 4 0 | END JK;
461 3 0 | END BRANCHMAD;
462 2 0 | /* PRINT SPECIFICATION-FUNCTION MATRIX */
463 3 0 | PPSPECM: PROCEDURE:
464 4 0 | /* GLOBAL: N,OTAB,N3,STAB */
465 4 0 | DCL J FIXED BINARY;
466 4 0 | /* RESET OTAB TO MEMORY ELEMENT COLUMN */
467 4 0 | DO J = 1,3 TO N+2;
468 4 0 | OTAB(J) = SUBSTR(OTAB(J),*,STAB);
469 4 0 | END;
470 3 0 | /* CONCATENATE OUTPUT MATRIX */
471 4 0 | CALL OUTMAT;
472 3 0 | /* OUTPUT MATRIX AFTER INSERTING BOUNDARIES */
473 4 0 | PUT PAGE LIST('CELL SPECIFICATION-FUNCTION MATRIX');
474 4 0 | PUT SKIP(4) EDIT('PRESENT',STATE, 'MEMORY ELEMENT PRIMARY OUTPUT',
475 4 0 | '||' (A,SKIP,A,SKIP,Y(N3),A);
476 4 0 | CALL PRINMAT;
477 3 0 | END PPSPECM;
478 2 0 | /* PRINT EXCITATION AND OUTPUT FUNCTIONS */
479 3 0 | PPSPEM: PROCEDURE:
480 4 0 | /* GLOBAL: N,STAB,N3,OTAB,PLINE */
481 4 0 | DCL (IS,IS,N3,N3,J,STOP,I,II,IJ) FIXED BINARY;
482 4 0 | DCL NEXTST BIT(1);
483 4 0 | PUT PAGE LIST('EXCITATION AND OUTPUT FUNCTIONS');
484 4 0 | PUT SKIP(2);
485 4 0 | N3 = N+3;
486 4 0 | J = STAB+2;
487 4 0 | STOP = LENGTH(OTAB(1))-2;
488 4 0 | DO WHILE (J < STOP);

```

STATE LEV "T

```

476 3 1 | BUT SKIP(2);
477 3 1 | CALL GETSTR(1);
478 3 1 | MAXLS = LS;
      | /* INITIALIZE PRINTLINE */
479 3 1 | BLINE = 0;
480 3 1 | CALL PRINTLINE(SUBSTR(OTAB(1),SS,LS) || '=',0);
481 3 1 | NFIRST = '012';
      | /* GET FUNCTION OF INPUT SYMBOLS */
482 3 1 | DO I = 3 TO 4+2;
483 3 2 |   CALL GETSTR(I);
484 3 2 |   IF SUBSTR(OTAB(I),SS,LS) <= '1' THEN
      |     DO;
485 3 3 |       IF NFIRST THEN
      |         CALL PRINTLINE('1',0);
486 3 3 |       ELSE
      |         NFIRST = '12';
      |       IF SUBSTR(OTAB(I),SS,LS) <= '1' THEN
      |         DO;
487 3 4 |           IF LS > MAXLS THEN
      |             MAXLS = LS;
488 3 4 |           IF INDEX(SUBSTR(OTAB(I),SS,LS),'1') > 0 THEN
      |             OTAB(N3) = '1' || SUBSTR(OTAB(I),SS,LS) || '1';
489 3 4 |           ELSE
      |             OTAB(N3) = SUBSTR(OTAB(I),SS,LS);
490 3 4 |         END;
491 3 4 |       ELSE
492 3 3 |         OTAB(N3) = SUBSTR(OTAB(N3),1,1);
      |       /* MULTIPLY BY FUNCTION OF STATE */
493 3 3 |       IS = I-2;
494 3 3 |       DO IU = 1 TO N;
495 3 4 |         IF NIGEST(I,IU) THEN
496 3 4 |           OTAB(N3) = OTAB(N3) || '1' || NIGEST(I,IU);
497 3 3 |       END;
498 3 3 |       CALL PRINTLINE(OTAB(N3),0);
499 3 3 |     END;
500 3 2 |   END;
501 3 1 |   J = J+MAXLS+1;
502 3 1 | END;
      | /* GET STRING STARTING AT J, FROM BEGINNING OF STRING IN SS,
      |    LENGTH IN LS = NO BLANKS */
503 3 1 | GETSTR: PROCEDURE (STEP);
      | /* GLOBAL: J,LS,SS,OTAB */
504 3 0 | DO (S,STEP) FIXED BINARY;
505 3 0 | SS = 0;
506 3 0 | DO WHILE (SUBSTR(OTAB(SS),SS,1) = ' ');
507 3 1 |   SS = SS+1;
508 3 1 | END;
509 3 0 | LS = 1;
510 3 0 | S = SS+1;
511 3 0 | DO WHILE (SUBSTR(OTAB(SS),S,1) <= '1');
512 3 1 |   S = S+1;
513 3 1 |   LS = LS+1;
514 3 1 | END;
515 3 0 | END GETSTR;

```


STMT LEV NT

```

515 3 0 | END PROCFUND;
516 2 0 | END;
      /* TEST N,W,M,U,V,K AND CALCULATE M,K IF NECESSARY */
      /* ALSO CALCULATE BOUNDS ON ZP AND ZL */
517 1 0 | CALCPRAM: PROCEDURE;
      /* GLOBAL: U,N,W,V,M,K,ZPY,ZBY,ZLX,ZLY */
      /* U,N MUST BE > 0 */
518 2 0 | IF U <= 0 THEN
      DO;
519 2 1 | PUT SKIP LIST('***ERROR*** U NOT IN RANGE, U=',U);
520 2 1 | STOP;
521 2 1 | END;
522 2 0 | IF N <= 0 THEN
      DO;
523 2 1 | PUT SKIP LIST('***ERROR*** N NOT IN RANGE, N=',N);
524 2 1 | STOP;
525 2 1 | END;
      /* W,V MUST BE >= 0 */
526 2 0 | IF W < 0 THEN
      DO;
527 2 1 | PUT SKIP LIST('***ERROR*** W NOT IN RANGE, W=',W);
528 2 1 | STOP;
529 2 1 | END;
530 2 0 | IF V < 0 THEN
      DO;
531 2 1 | PUT SKIP LIST('***ERROR*** V NOT IN RANGE, V=',V);
532 2 1 | STOP;
533 2 1 | END;
      /* W+V MUST BE >= 1 */
534 2 0 | IF W = 0 & V = 0 THEN
      DO;
535 2 1 | PUT SKIP LIST('***ERROR*** NO OUTPUTS SPECIFIED, W+V=0');
536 2 1 | STOP;
537 2 1 | END;
      /* M MUST BE >= 0 */
538 2 0 | IF M < 0 THEN
      DO;
539 2 1 | PUT SKIP LIST('***ERROR*** M NOT IN RANGE, M=',M);
540 2 1 | STOP;
541 2 1 | END;
      /* K MUST BE >= -1 */
542 2 0 | IF K < -1 THEN
      DO;
543 2 1 | PUT SKIP LIST('***ERROR*** K NOT IN RANGE, K=',K);
544 2 1 | STOP;
545 2 1 | END;
      /* CALCULATE M,K IF NECESSARY */
      /* ARE M & K SPECIFIED? */
546 2 0 | IF M > 0 & K >= 0 THEN
      DO;
      IF C(M,K) < M THEN
547 2 1 | PUT SKIP LIST('***ERROR*** M/(M*(M-K)) < 1');
548 2 1 | PUT SKIP LIST('M,K,M');
549 2 1 | STOP;

```

STATE LEV NO

```

550 2 1 |      END;
551 2 0 |      ELSE;
552 2 0 |      ELSE
/* M SPECIFIED, K NOT? */
      IF M > 0 & K = -1 THEN
        DO;
553 2 1 |          DO K = 0 TO FLOOR(M/2);
554 2 2 |              IF C(M,K) >= N THEN
                    RETURN;
555 2 2 |          END;
556 2 1 |          PUT SKIP LIST('***ERROR*** M NOT LARGE ENOUGH, M=',M);
557 2 1 |          STOP;
558 2 1 |      END;
559 2 0 |      ELSE
/* K SPECIFIED, M NOT? */
      IF K >= 0 THEN
        DO M = K BY 1 WHILE (C(M,K) < N);
560 2 1 |      END;
561 2 0 |      ELSE
/* M & K NOT SPECIFIED */
        DO;
562 2 1 |          DO M = 1 BY 1 WHILE (C(M,FLOOR(M/2)) < N);
563 2 2 |          END;
564 2 1 |          DO K = FLOOR(M/2) BY -1 WHILE (C(M,K) > N);
565 2 2 |          END;
566 2 1 |          IF C(M,K) < N THEN
                    K = K + 1;
567 2 1 |          END;
/* CALCULATE BOUNDS FOR ZP */
568 2 0 |      IF W = 0 THEN
        DO;
569 2 1 |          ZPX = 1; ZPY = 1;
570 2 1 |          END;
571 2 0 |      ELSE
        DO;
572 2 0 |          ZPX = N; ZPY = 0 * W;
573 2 1 |          END;
574 2 1 |      /* CALCULATE BOUNDS FOR ZI */
      IF V = 0 THEN
        DO;
575 2 1 |          ZLX = 1; ZLY = 1;
576 2 0 |          END;
577 2 0 |      ELSE
        DO;
578 2 1 |          ZLX = N; ZLY = V;
579 2 1 |          END;
580 2 0 |      END CALCOPAFM;
/* CALCULATE M CHOOSE K */
581 1 0 |      PROCEDURE (M,K) RETURNS (FIXED BINARY);
582 2 0 |      DCL (M,K,ANS,B,I) FIXED BINARY;
583 2 2 |      IF K > M THEN
        RETURN (0);
584 2 0 |      IF K = 0 | K = M THEN
        RETURN (1);

```

PL/I OPTIMIZING COMPILER

PAROSSM: PROCEDURE OPTIONS (MAIN);

STMT LEV NT

```

588 2 0 | IF K = 1 | F = V - 1 THEN
589 2 0 | RETURN (M);
590 2 0 | ANS = 1;
591 2 0 | B = MIN(K,M-K);
592 2 0 | DO I = 0 TO B - 1;
593 2 1 | ANS = ANS * (M - I) / (B - I);
594 2 1 | END;
595 2 0 | RETURN (ANS);
596 2 0 | END C;
/* CONVERT INTEGER TO CHARACTER, SUPPRESSING LEADING BLANKS */
597 1 0 | NUMCHAF: PROCEDURE (I) RETURNS (CHARACTER (5) VARYING);
598 2 0 | DCL I FIXED BINARY;
599 2 0 | DCL NUMCH CHARACTER (5) VARYING;
600 2 0 | PUT STRING (NUMCH) EDIT (I) (F(5));
601 2 0 | DO WHILE (INDEX(NUMCH,' ') /= '');
602 2 1 | NUMCH = SUBSTR(NUMCH,2);
603 2 1 | END;
604 2 0 | RETURN (NUMCH);
605 2 0 | END NUMCHAF;
606 1 0 | END PAROSSM;

```

1.

MAC-10 14 1075.0:

FLIP FLOP LIST:

STATE Table

2-00-CF-4 STAFF ASSISTANT

PA-TITN (C) 2013-2014

DESSERT

	1	51	52	1
01	01.0	05.0	1	0
02	01.0	06.0	1	1
03	02.0	04.1	1	
04	03.0	07.0	1	
05	02.0	04.1	1	1

PRESIDENT

	V_1	V_2	V_3	V_4
Q1	1	1	0	0
Q2	1	1	1	0
Q3	1	1	0	1
Q4	1	0	1	1
Q5	1	0	1	0

011-14-732,2,1)

$$P(1) = (2, 7, 1, 0, 0, \dots)$$
$$O_1^* = (1, 1, \dots, 1, -1)$$
$$D_1 = (1, 2, \dots, 7, 8)$$

CPY:EC71.0. 147512

OUTPUT 22-18

AFI 10-671, 1987

2025

	1	01	02	03	04	05
01	1	51	0	0	0	02
02	1	51	0	51	0	0
03	1	0	51	0	02	0
04	1	0	0	51+52	0	0
05	1	0	0	51	52	0

087535.T

	1	201	211	
01	1	51	0	0
02	1	0	1	0
03	1	87	0	1
04	1	51	0	1
05	1	0	1	1

2. 2. 2. 2. 2

	1	2	3	4	5	6
21	1	2	3+3	4	5	6
22	1	1+2	3	4	5	6
23	1	2	3	4+5	6	7
24	1	3+2	4	5	6+7	8
25	1	2	3	4	5	6+7

CELL SPECIFICITY - NOTED THAT IN

EXCERPT FROM THE 1977-1978 REPORT

ORIGINAL 100-443887-100

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
01	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
02	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
03	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
04	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
05	1	2	3	4	5																																																																																															

[illegible]
$$Y^2 = (X^2 + 5)(X^2 + 1)(X^2 + 2) = (X^2 + 5)(X^2 + 1)(X^2 + 2) = (X^2 + 5)(X^2 + 1)(X^2 + 2).$$
$$y_3 = (y_1 + y_2) \vee (x_1 + x_2) \vee y_4$$
$$y_{it} = \beta_0 + \beta_1 y_{it-1} + \beta_2 y_{it-2} + \beta_3 y_{it-3} + \beta_4 y_{it-4} + \beta_5 y_{it-5}$$
$$f'(x) = 5x^4 + 3x^2 + 1$$
$$Z(t) = Y(t) + \varepsilon(t)$$

SECTION III

MATHEMATICAL MODELS FOR THE
DESIGN AND ANALYSIS OF
ON-LINE BUILT-IN-TESTING

by

Kishor S. Trivedi
Computer Science Department
Duke University

January 1979

1.0 INTRODUCTION

This paper is concerned with the analysis and design of on-line Built-In-Test (BIT). Such systems are characterized by on-line fault monitoring, and therefore, a study of the effectiveness of on-line fault monitors is important [1,2]. Existing models of systems analysis [3,4] are inadequate for modeling systems with on-line fault monitoring since they assume that fault detection occurs in zero time and that the fault monitor never fails. Our models will allow a finite detection latency, an imperfect fault monitor, multiple fault monitors, and multiple classes of faults.

The analysis problem occurs when the system structure is specified and we are interested in evaluating the performance of the system. Such an analysis will be probabilistic in nature since the failure modes of various components of the system are probabilistic. If a repair facility is included in our model, (i.e., the system is repairable or maintained), then the performance measure of interest is the steady state system availability. If the system is non-maintained or has imperfect coverage [5,6], the performance measure of interest is the system reliability as a function of the mission time.

Due to finite detection latency, it is possible that a fault has occurred in the system but it is not yet detected [7]. Such a state of the system is clearly undesirable, since erroneous outputs produced during this state may propagate and contaminate system data bases so as to make recovery extremely difficult, if not impossible [8]. The purpose of an on-line fault monitor (detector) is to reduce the probability that the system is in the undesirable state. We will give explicit expressions for the effectiveness of a fault-monitor in achieving this goal.

It is also possible to improve the reliability of fault-tolerant schemes such as TMR, NMR, or hybrid-NMR [4] by employing an on-line detector with each module. Such systems have been called DR (detector redundant) by Ramamoorthy and Han [9], SCR (self-checking redundant) by Karavay and Sogomonyan [10], and self-purging redundant by Losq [11]. The use of on-line fault detectors overcomes the inherent complexity of the switches employed in the fault-tolerant schemes above and also provides fault masking capability for multiple fault occurrence. For the analysis of the above schemes, the reader is referred to [9,10,11].

The problem of system design is to configure the optimal system for the stated purpose. In our context, we are interested in choosing a fault monitor that yields a system with optimum cost performance. The trade-off is between the cost of the monitor and the cost associated with the time the system spends in the undesirable state. In several simple cases, we will give closed form solutions that characterize the optimal fault monitor.

The basic constituent of the system that we consider is called a module and is shown in Figure 2.1. Two types of modules will be considered. One type is the non-maintained (or non-reparable) module M and the other type is the maintained (or reparable) module M'. Module M consists of the functional unit U and its on-line fault detector D. The module M' consists of the functional unit U, the detector D and a repair facility R. An example of a functional unit is an arithmetic unit, and the corresponding detector could be its modulo-3 checker. If the functional unit is a complex processing unit, then the detector could be a software routine executed on a microprocessor.

2.0 ANALYSIS

2.1 Analysis of a Maintained Module

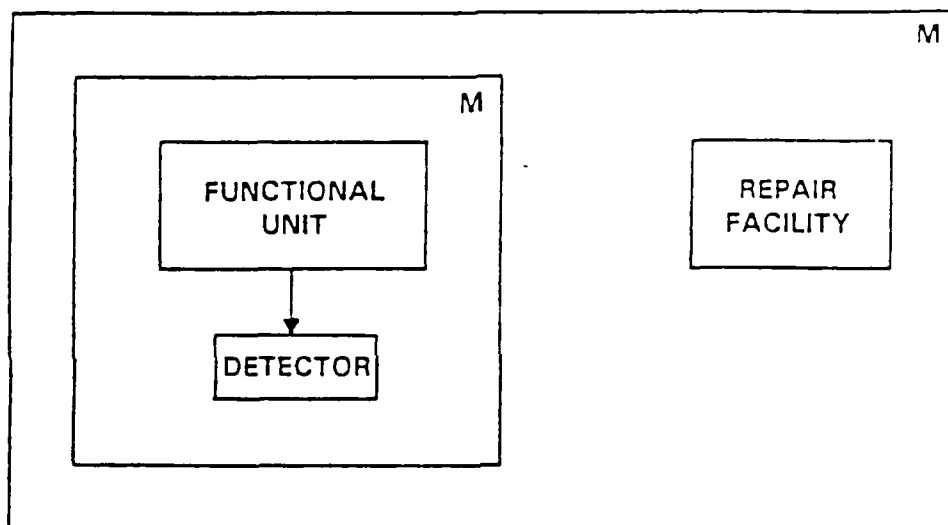
We will first present the well-known analysis of a simple maintained module, devoid of the on-line detector D. Throughout the subsequent discussion we will assume that the time between two successive failures of the functional unit U is exponentially distributed with mean $1/\lambda$. Thus, the failure rate is λ and the Mean-Time-Between-Failures (MTBF) is $1/\lambda$. We assume that the time to repair is exponentially distributed with mean $1/\mu$. Thus, the repair rate is μ and the Mean-Time-To-Repair (MTTR) is $1/\mu$.

The module has two possible states, F (failed) and W (working properly). The state diagram of the module is shown in Figure 2.2. Let P_F be the steady state probability that the system is in state F. It can be shown that [3]

$$P_W = \frac{\mu}{\lambda + \mu}$$

and

$$P_F = \frac{\lambda}{\lambda + \mu}$$



M : NON-MAINTAINED MODULE
M': MAINTAINED MODULE

Figure 2.1 Basic System Module

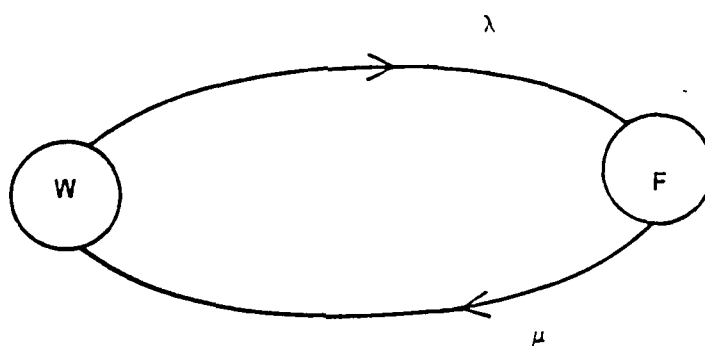


Figure 2.2 A Two-State Model

Now, the module availability A is simply P_W by definition. Thus

$$A = \frac{\mu}{\lambda + \mu} = \frac{1/\lambda}{1/\lambda + 1/\mu} = \frac{MTBF}{MTBF + MTTR} \quad (3)$$

There are many drawbacks of this simple well-known model of availability. First, it assumes that the detector is perfect; i.e., the detector never fails. Second, it assumes that the time to detect failures is negligible or the detection latency is zero. We present a model below that removes both these drawbacks.

We make all the assumptions made earlier for the simple two-state model. In addition, we assume that the time to detect failures is exponentially distributed with mean $1/\delta$. Thus, the detection rate is δ and the Mean Time-To-Detect-Failures (MTDF) is $1/\delta$. The time to failure and the time to repair for the detector are exponentially distributed with mean $1/\alpha$ and $1/\beta$, respectively where α may also be interpreted as the false alarm rate. The module can be in any one of the four states: W, F, D and C. In state W, the module is functioning properly; in state F, the functional unit has failed but the failure is not yet detected. In state D, the failure is detected and the functional unit is under repair. In state C, the detector has failed and it is under repair. The state diagram is given in Figure 2.3. The steady state probabilities for each of these states can be obtained as

$$P_W = \frac{1}{1 + \lambda/\mu + \lambda/\delta + \alpha/\beta},$$

$$P_F = \frac{\lambda/\delta}{1 + \lambda/\mu + \lambda/\delta + \alpha/\beta},$$

$$P_D = \frac{\lambda/\mu}{1 + \lambda/\mu + \lambda/\delta + \alpha/\beta} \text{ and}$$

$$P_C = \frac{\alpha/\beta}{1 + \lambda/\mu + \lambda/\delta + \alpha/\beta}.$$

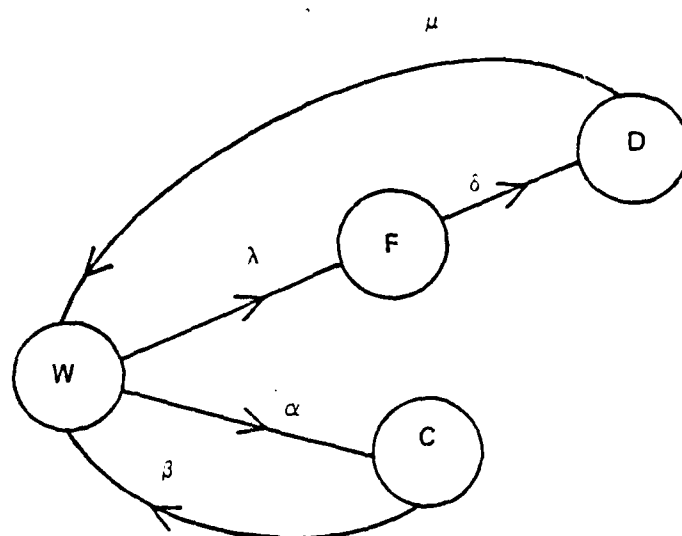


Figure 2.3 A Four-State Model

It is interesting to observe that when the module is in state F, it has malfunctioned but the outside world does not know it. Thus, from an external point of view the module is said to be available when it is either in state W or in state F. In reality, the module should be called available only when it is in state W. Thus, we have the real availability $A_r = P_W$ and the apparent availability $A_a = P_W + P_F$. The purpose of the on-line detector is to keep A_a and A_r as close to each other as possible.

Note that the real availability

$$\begin{aligned} A_r &= \frac{1}{1 + \frac{\lambda}{\mu} + \frac{\lambda}{\delta} + \frac{\alpha}{\beta}} = \frac{1}{1 + \lambda \left(\frac{1}{\mu} + \frac{1}{\delta} \right) + \alpha/\beta} \\ &= \frac{1}{1 + \frac{MTTR + MTDF}{MTBF} + \frac{\alpha}{\beta}} \end{aligned}$$

Now, since α is usually much smaller than λ , we may let

$$A_r \approx \frac{MTBF}{MTBF + (MTTR + MTDF)} \quad (4)$$

Comparing expression (4) with expression (3), we conclude that $MTTR + MTDF$ behave like an "effective" repair time. The use of a more powerful detector, i.e., a smaller value of $MTDF$, implies a reduction in the effective repair time, which in turn implies an increase in real availability. In fact, for fixed values of $MTBF$ and $MTTR$, largest real availability results when we employ a detector with zero detection latency.

Next, consider the probability of being in the undesirable state F

$$\begin{aligned} P_F &= \frac{\lambda/\delta}{1 + \lambda/\mu + \lambda/\delta + \alpha/\delta} \\ &\approx \lambda/\delta \left(1 - \lambda/\mu - \lambda/\delta - \alpha/\delta \right) \\ &\approx \frac{\lambda}{\delta} \left(1 - \lambda/\mu - \frac{\alpha}{\delta} \right) - \frac{\lambda^2}{\delta^2} \\ &\approx \frac{\lambda}{\delta} \left(1 - \lambda/\mu - \frac{\alpha}{\delta} \right) \end{aligned} \quad (5)$$

thus employing a more powerful detector (i.e., increasing the value of δ), reducing P_F , and hence, bringing the real availability A_r and the apparent availability A_a closer together. In fact, a detector with an infinite detection rate (or zero detection latency) implies that $P_F = 0$ and $A_a = A_r$. In this case, there is no need to distinguish between the concepts of real and apparent availabilities.

Finally, consider the apparent availability

$$\begin{aligned}
 A_a &= P_W + P_F \\
 &= \frac{1 + \lambda/\delta}{1 + \frac{\lambda}{\mu} + \frac{\lambda}{\delta} + \frac{\alpha}{\beta}} \\
 &= 1 - \frac{\lambda/\mu + \alpha/\beta}{1 + \lambda/\mu + \lambda/\delta + \alpha/\beta} \\
 &= 1 - \left(\frac{\lambda}{\mu} + \frac{\alpha}{\beta}\right) A_r
 \end{aligned} \tag{6}$$

Now, since A_r increases with an increase in δ (i.e., a more powerful detector), we conclude that the apparent availability reduces with an increase in δ . Thus, A_a and A_r approach each other as δ increases and $A_a = A_r$ in the limit $\delta \rightarrow \infty$. Further analysis suggests that the rate of decrease in A_a is very slow since

$$\begin{aligned}
 A_a &\approx 1 - \left(\frac{\lambda}{\mu} + \frac{\alpha}{\beta}\right) (1 - \lambda/\mu - \lambda/\delta - \alpha/\beta) \\
 &\approx 1 - \frac{\lambda}{\mu} - \frac{\alpha}{\beta}
 \end{aligned} \tag{7}$$

Thus, as a first order approximation, the apparent availability remains constant, independent of the mean detection latency.

To fix our ideas, let $\lambda = 10^{-5}/\text{hr}$, $\mu = 2/\text{hr}$, $\alpha = 10^{-6}/\text{hr}$, and $\beta = 4/\text{hr}$. In Figure 2.4 we have plotted the apparent availability A_a and the real availability A_r as functions of MTDF.

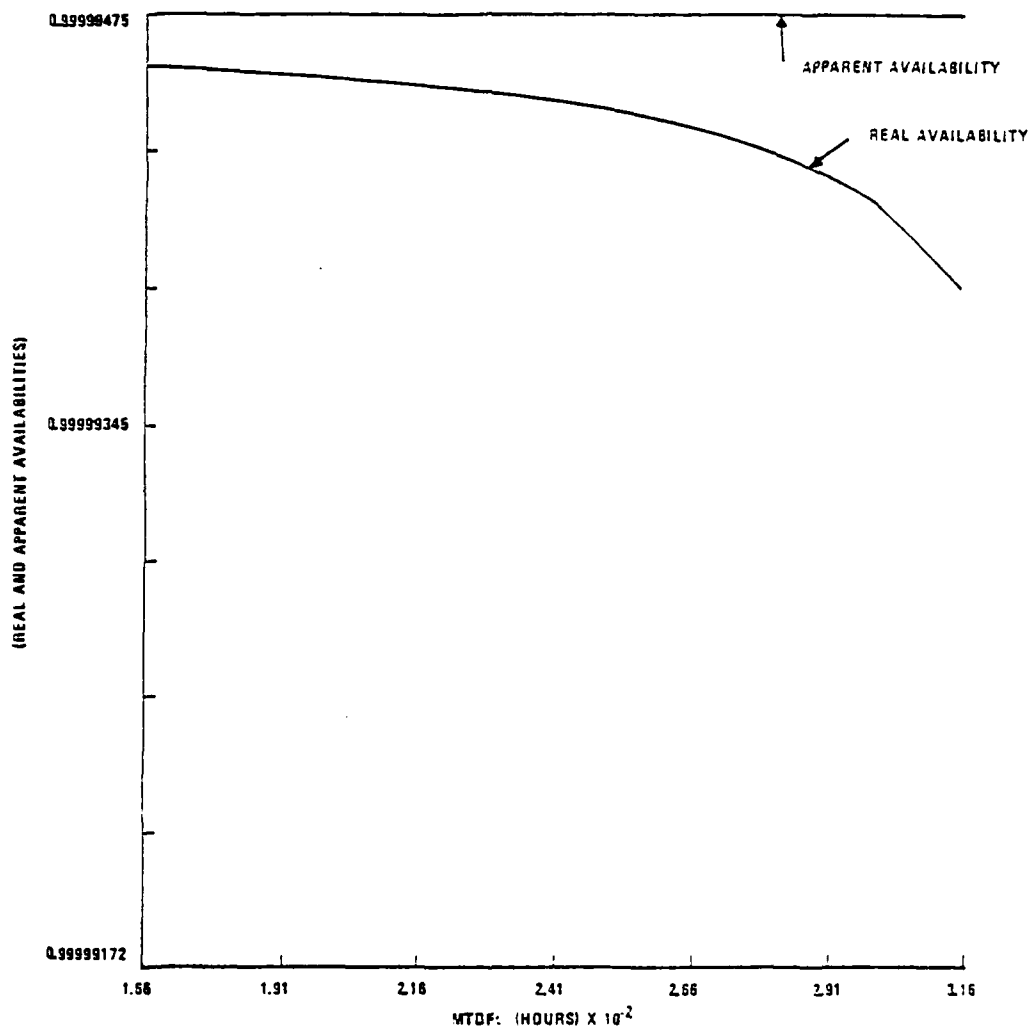


Figure 2.4 Apparent Availabilities vs MTDF

2.2 Multiple Fault Classes

In general, different types of faults will require different amounts of time for detection and repair. We assume there are k fault classes each with a failure rate λ_i , detection rate δ_i , and repair rate μ_i . Figure 2.5 is a state diagram for the Markov chain of the system.

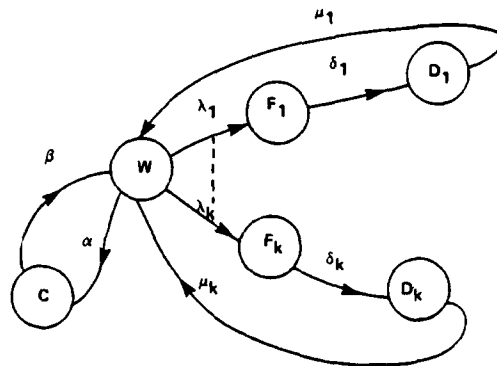


Figure 2.5

The steady state probabilities for being in each of state, λ , can be given as,

$$P_W = \frac{1}{1 + \left(\sum_{i=1}^k \frac{\lambda_i}{\mu_i} \right) + \sum_{i=1}^k \left(\frac{\lambda_i}{\delta_i} \right) + \frac{\alpha}{\beta}}$$

$$P_{F_i} = \frac{\lambda_i}{\delta_i} P_W, P_{D_i} = \frac{\lambda_i}{\mu_i} P_W, P_C = \frac{\alpha}{\beta} P_W$$

The real availabilities

$$\begin{aligned} A_r = P_W &= \frac{1}{1 + \sum_{i=1}^k \lambda_i \left(\frac{1}{\delta_i} + \frac{1}{\mu_i} \right) + \frac{\alpha}{\beta}} \\ &= \frac{1}{1 + \sum_{i=1}^k \frac{\text{MTTR}_i + \text{MTDF}_i}{\text{MTBF}_i} + \frac{\alpha}{\beta}} \end{aligned}$$

Note that each failure class gives a contribution to a reduction in A_r . Thus, in effect, these classes are in "series".

2.3 Multiple Detectors

It is clear that we can improve the detectability of a module by employing several detectors in parallel. Assume there is a single fault class with failure rate λ . There are k detectors with detection rate δ_i and the associated repair rate is μ_i . The state diagram is shown in Figure 2.6.

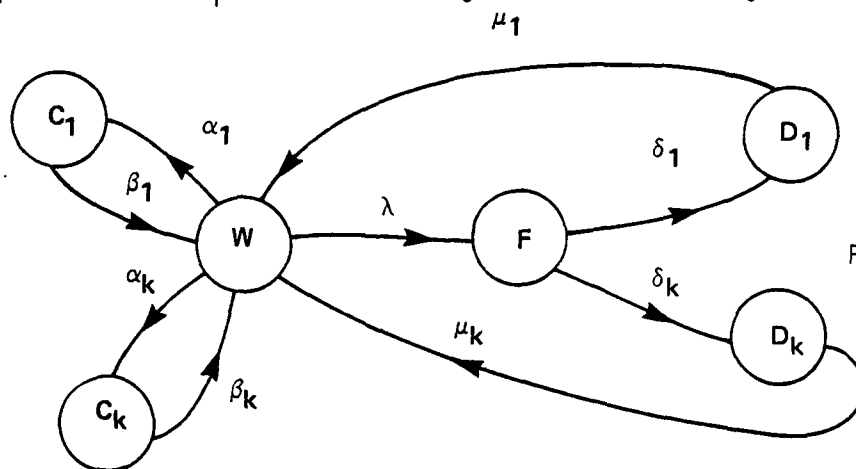


Figure 2.6

$$A_r = P_w = \frac{1}{1 + \sum_{i=1}^k \frac{\alpha_i}{\beta_i} + \frac{1}{\frac{\lambda}{k} \left(1 + \sum_{i=1}^k \frac{\delta_i}{\mu_i} \right)}}$$

If we assume that $\mu_i = \mu$ for all i , then

$$\begin{aligned} A_r &= \frac{1}{1 + \sum_{i=1}^k \frac{\alpha_i}{\beta_i} + \frac{1}{\left(\frac{1}{k} + \frac{1}{\mu} \right)}} \\ &= \frac{1}{1 + \frac{\text{MTTR} + \text{MTDF parallel}}{\text{MTBF}} + \sum_{i=1}^k \frac{\text{MTTR}_{Di}}{\text{MTBF}_{Di}}} \end{aligned}$$

In other words, the detection rate of the detectors have an additive effect and the result is an increased real availability. But increased number of detectors also imply an increased false alarm rate which tends to decrease the real availability.

Assuming $\lambda = 10^{-5}/\text{hr.}$, $\mu = 2/\text{hr.}$, and $\alpha = 0$, the following table 2.1 gives the real availability for various combinations of Modulo-m detectors for the ALU shown in Figure 2.7.

Table 2.1 Various Combinations of Modulo-m Detectors

M.		One Detector			Two Detectors			Three Detectors
		3	5	7	3 + 5	3 + 7	5 + 7	3 + 5 + 7
Detection Rates :	δ_i	.66	.80	.86	.66 + .80	.66 + .86	.80 + .86	.66 + .80 + .86
Real Availability:	Ar	.99997960	.99998225	.99998315	.99998765	.99998792	.99998848	.99998994
Unavailability:		$.204 \times 10^{-4}$	0.1775×10^{-4}	0.1685×10^{-4}	0.1235×10^{-4}	0.1208×10^{-4}	0.1152×10^{-4}	0.1006×10^{-4}

The data on the detection rate δ for various values of m was obtained from [12]. It is clear that real availability increases with an improved detector and it also increases by employing multiple detectors.

2.4 Analysis of a Non-Maintained Module

We will now consider the analysis of a non-maintained module M consisting of the functional unit U and the associated detector (or fault monitor) D . Let U and C denote the time to failure of the unit and the detector, respectively. Let D be the time to detect a fault in the module. Let T denote the time to fault indication. Note that U , C , D and T are all random variables and $T = \min(U + D, C)$. Let $R_a(t)$ and $R_r(t)$ denote the apparent and the real reliabilities of the module, in the order given. We will assume that U , D and C are mutually independent, exponentially distributed random variables with means $1/\lambda$, $1/\delta$ and $1/\alpha$, respectively. Then

$$\begin{aligned}
 R_a(t) &= P(T > t) = P(U + D > t, C > t) \\
 &= P(U + D > t) P(C > t) \quad \text{by independence} \\
 &= R_{U+D}(t) R_C(t)
 \end{aligned} \tag{8}$$

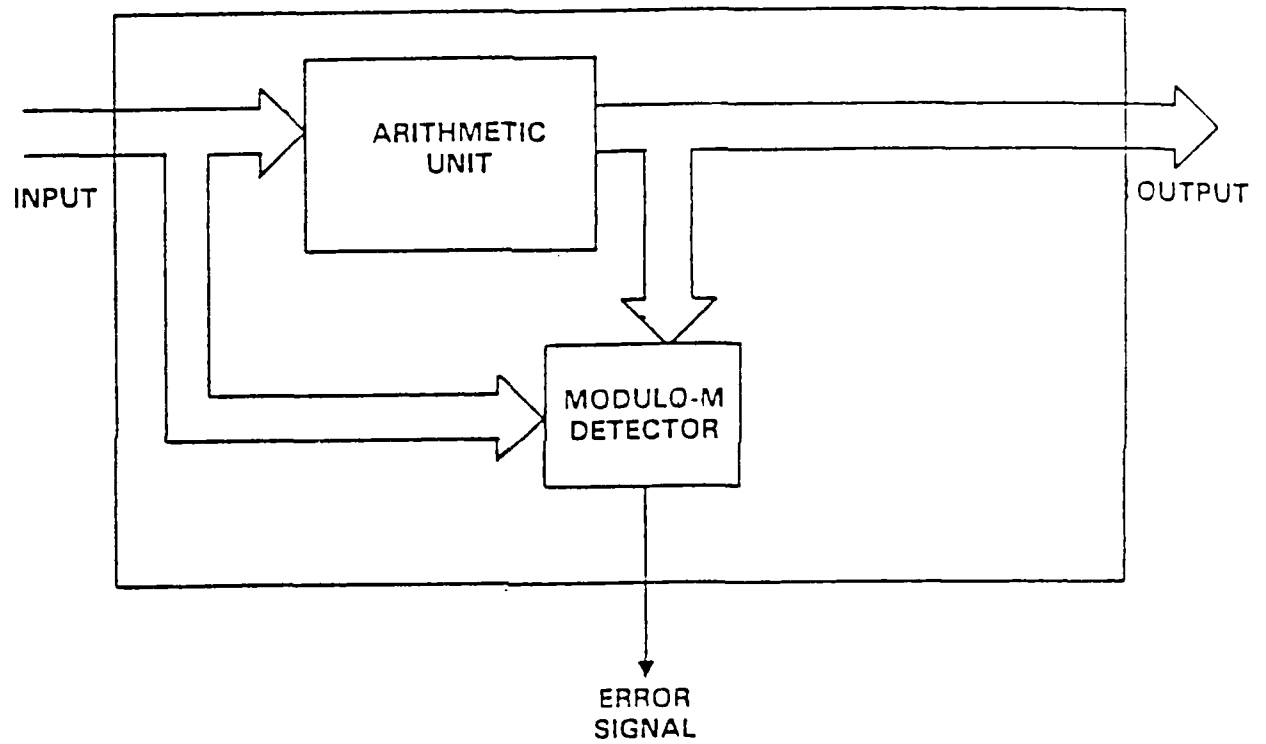


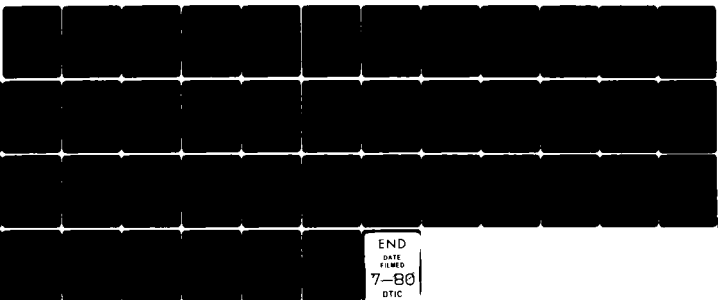
Figure 2.7 Design of a Non-Maintained Module

AD-A004 808

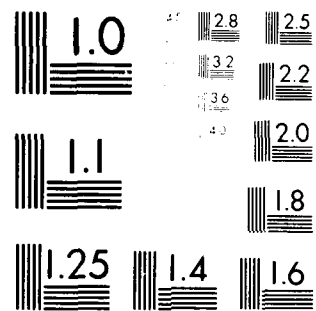
RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/O 12/1
TECHNIQUES FOR ON-LINE FAULT MONITORING IN MODULAR DIGITAL SYST--ETC(U)
JAN 80 J W SAULT, K S TRIVEDI, P N MARINOS N00039-78-C-0031
RTI/1667/00-01F NL

UNCLASSIFIED

4 of 4
AD-A 004 808



END
DATE
FILMED
7-80
DTIC



MICROCOPY RESOLUTION TEST CHART

ANSI #2 - 1963-A

By exponential assumption,

$$R_C(t) = e^{-\alpha t}, R_D(t) = e^{-\delta t}, \text{ and } R_U(t) = e^{-\lambda t}.$$

Therefore,

$$R_{U+D}(t) = \frac{\delta}{\delta - \lambda} e^{-\lambda t} - \frac{\lambda}{\delta - \lambda} e^{-\delta t} \quad (9)$$

Then, from the above, we get

$$R_a(t) = \frac{\delta}{\delta - \lambda} e^{-(\lambda+\alpha)t} - \frac{\lambda}{\delta - \lambda} e^{-(\delta+\alpha)t} \quad (10)$$

For computing real reliability $R_r(t)$, we note that the module ceases to function properly when a fault occurs in either the unit or the detector.

Thus,

$$\begin{aligned} R_r(t) &= P(U > t, C > t) \\ &= P(U > t) P(C > t) \quad \text{by independence} \\ &= R_U(t) R_C(t) \\ &= e^{-(\lambda+\alpha)t} \end{aligned} \quad (11)$$

We note that, in the absence of the detector, the real reliability is $e^{-\lambda t}$; therefore, employing a detector actually reduces the real reliability.

Without a detector, $\alpha = 0$, δ is near zero, and the apparent reliability is very high. Thus, the apparent reliability is also reduced by employing a detector. The purpose of an on-line detector is to close the gap between the values of the real and the apparent reliabilities.

We can also compute the real and apparent MTTF ($MTTF_r$ and $MTTF_a$):

$$MTTF_r = \frac{1}{\lambda + \alpha} \quad (12)$$

and

$$\begin{aligned} MTTF_a &= \frac{\delta}{(\delta - \lambda)(\lambda + \alpha)} - \frac{\lambda}{(\delta - \lambda)(\delta + \alpha)} \\ &= \frac{\delta + \lambda + \alpha}{(\lambda + \alpha)(\delta + \alpha)} \end{aligned} \quad (13)$$

We define the detector effectiveness to be the ratio $MTTF_r/MTTF_a$. The detector effectiveness is plotted in Figure 3.1 as a function of the detection rate δ .

3.0 DESIGN

We now present a design model for a non-maintained module. We are asked to choose the characteristics of an on-line detector that will minimize the total cost. The two cost components that enter into our model are the cost of the detector and the cost (or penalty) for the time system spent in the undesirable state. For the sake of simplicity, we will assume that the detector has zero failure rate, i.e., $\alpha = 0$.

The percentage of time spent by the module in the undesirable state is easily computed to be λ/δ . Let the per unit time penalty of being in this state be given by K_F . Then, the penalty is given by $K_F \lambda/\delta$. To characterize the cost of the detector, we assume that the unit U is an arithmetic unit and the detector D is a modulo- m checker (see Figure 3.1). The problem, then, is to determine the optimum value of m .

The cost of a modulo- m checker may be approximated by $C_0 \log m$. Then the total cost

$$C = K_F \lambda/\delta + C_0 \log m \quad (14)$$

Note that λ , K_F and C_0 are assumed to be fixed parameters, but δ is expected to be a function of m .

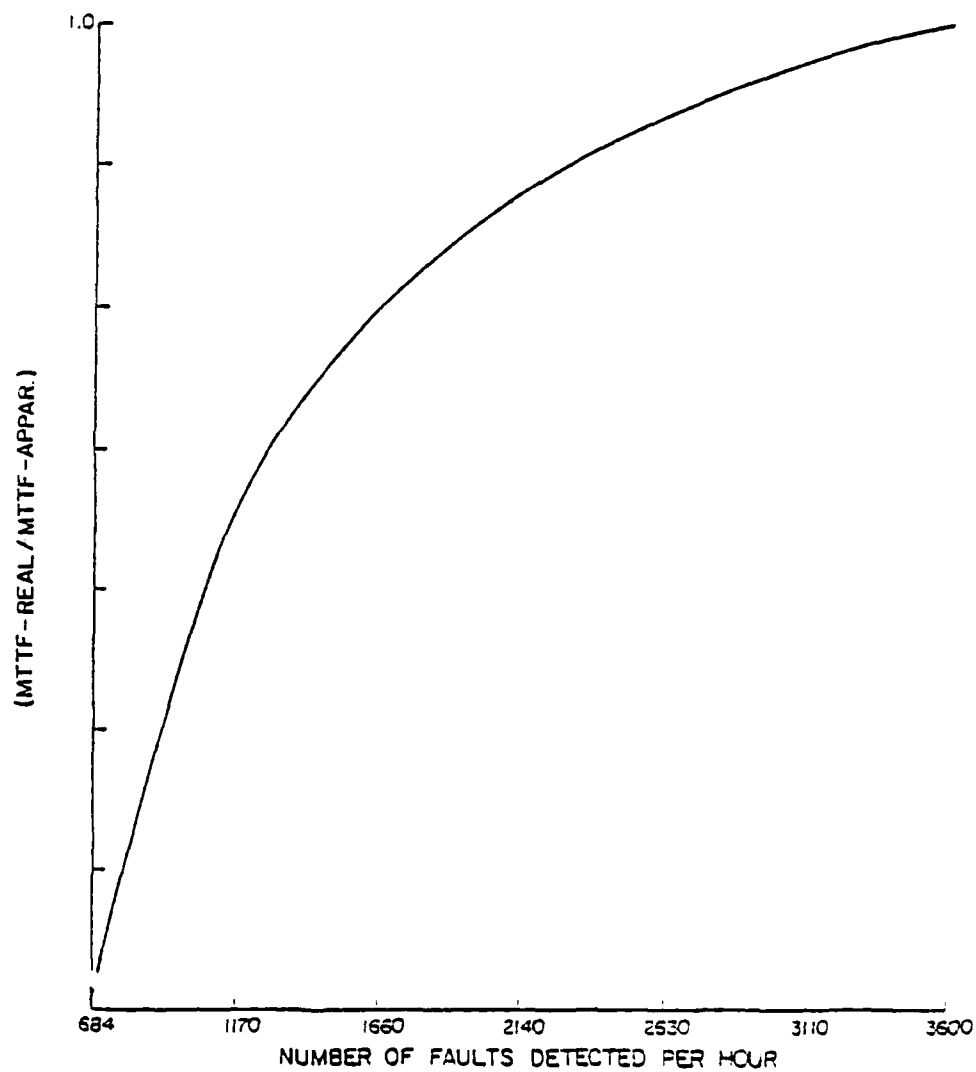


Figure 3.1 Detector Effectiveness

A reasonable functional relationship is

$$\delta = \delta_0 m^a \quad (15)$$

After substituting (15) into (14), we can determine the optimum value of m by taking $\frac{dC}{dm}$ and setting it equal to zero

$$\frac{dC}{dm} = - \frac{a K_F \lambda}{\delta_0 m^{a+1}} + \frac{C_0}{m} = 0 \quad (16)$$

or

$$m_{opt} = \sqrt{\frac{K_F}{C_0} \cdot \frac{\lambda a}{\delta_0}}$$

This shows that the larger the value of K_F relative to C_0 , the larger should be the value of m . In other words, if the penalty of being in the undesirable state is large, we should choose a more powerful detector.

It is hard to parameterize such a model. We obtained data from a paper by J. Clary [12] and fitted the data to obtain the values of δ_0 and a . Using these values and $\lambda = 10^{-5}/\text{hr}$, we can determine the optimal value of m as a function of the relative cost K_F/C_0 from equation (16). This function is plotted in Figure 3.2.

4.0 WORK PLANNED FOR THE FUTURE

We believe that models developed during the last two years are capable of evaluating the effectiveness of conventional concurrent on-line BIT. The projected work can be divided into three disjoint categories.

First, although some examples of the use of the models are given, more extensive examples are desired. Some extensions of the models developed so far are also possible, e.g., design models for maintained modules.

The second category of our effort will be to adapt our models to statistical monitoring approach studied by J. Gault [13]. This effort will revolve around computation of the probability of false alarm and the probability of escape. We are interested in compacting the two measures into a single measure such as reliability or availability. The basic difference between the models

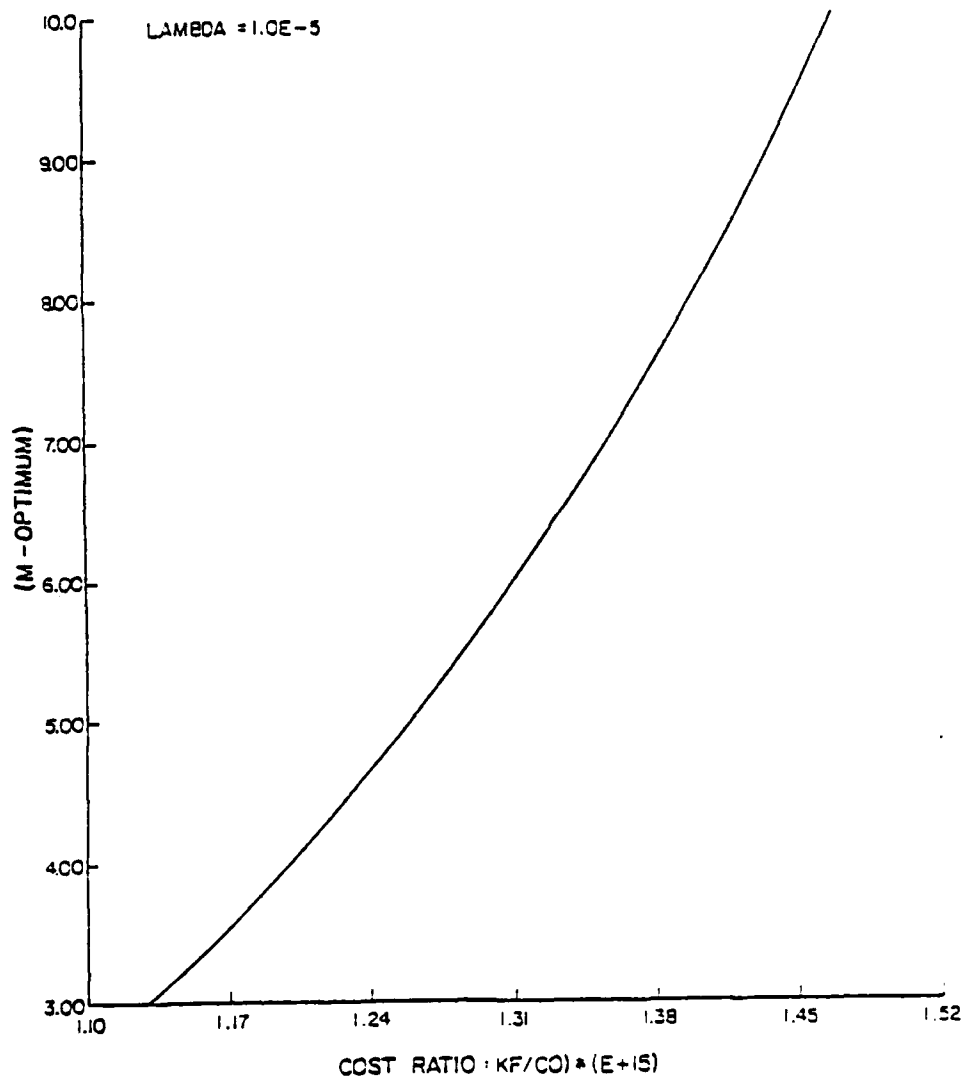


Figure 3.2 Optimum Check Modulus

developed by this researcher and models needed for sample fault monitoring is that the former models give rise to a continuous parameter Markov chain; whereas, the latter will give rise to a discrete parameter Markov chain. The task then will be to transform the former models to a discrete rather than a continuous time environment.

The third category of our effort will involve characterizing the fault-detecting and fault-correcting capability of the programmable cellular structure developed by P. N. Marinos [14]. Given a fixed cellular structure, we are interested in developing models that provide a measure of effectiveness of the structure and its self-checkability. Similarly, we are interested in characterizing the trade-offs involved in allocating a fraction of the given modules to the checking function.

The last two parts of our proposed research will therefore attempt to coordinate the efforts of the three different researchers working on the project.

REFERENCES

1. Clary, J. B., Gault, J. W., Weikel, S. J., Whisnant, R. A., Alberts, R. D., "A Study of a Standard BIT Circuit", Final Report Contract N00163-76-C-0231, Research Triangle Institute, Research Triangle Park, N.C., 1977.
2. Clary, J. B., Gault, J. W., Marinos, P. N., Trivedi, K. S., Whisnant, R. A., and Alberts, R. D., "Basic Research in Support of Concurrent Fault Monitoring in Modular Digital Systems", Proposal, Contract No. N00039-77-PR-7J-133, Research Triangle Institute, Research Triangle Park, N.C., 1977.
3. Barlow, R. E., and Proschan, F., Statistical Theory of Reliability and Life Testing: Probability Models, Holt, Rinehart and Winston, New York, 1975.
4. Mathur, F. P., and Avizienis, A. A., "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair", in 1970 Spring Joint Computer Conf., AFIPS Conf. Proc., Vol. 36. Montvale, N.J.: AFIPS press, 1970, pp. 375-383.
5. Bouricius, W. G., Carter, W. C., and Schneider, P. R., "Reliability Modeling Techniques for Self-Repairing Computer Systems", in Proc. 1969 Ann. Ass. Comput. Mach. Conf., pp. 295-309.
6. Arnold, T. F., "The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System", IEEE Trans. on Comput., Vol. C-22, No. 3, March 1973, pp. 251-254.
7. Shedletsky, J. J., "A Rollback Interval for Networks with an Imperfect Self-Checking Property", IEEE Trans. on Comput., Vol. C-27, No. 6, June 1978, pp. 500-508.
8. Toy, W. N., "Fault-Tolerant Design of Local ESS Processors", Proc. IEEE, Vol. 66, No. 10, October 1978, pp. 1126-1145.
9. Ramamoorthy, C. V., and Han Y. W., "Reliability Analysis of Systems with Concurrent Error Detection", IEEE Trans. on Comput., Vol. C-24, No. 9, September 1975, pp. 868-878.
10. Karavay, M. F., and Sogomonyan, E. S., "Compared Reliability Analysis of Redundant Systems", Proc. The Eighth Annual Conference on Fault-Tolerant Computing, p. 222, 1978.
11. Losq, J., "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy", IEEE Trans. Comput., Vol. C-25, No. 6, June 1976, pp. 569-578.
12. Clary, J. B., "Effectiveness Measures for Built-In-Test Performance Evaluation", Technical Report, Research Triangle Institute, Research Triangle Park, N.C., 1977.

13. Gault, J., "Sample Fault Monitoring" in "Basic Research in Support of Concurrent Fault Monitoring", RTI Report, June 1978.
14. Marinos, P. N., "BIT Facility Identification and Evaluation", in "Basic Research in Support of Concurrent Fault Monitoring", RTI Report, June 1978.

SECTION IV

A FLEXIBLE FAULT-TOLERANT
MULTIPROCESSOR ARCHITECTURE FOR
VIDEO GRAPHICS

by

Henry Fuchs
Computer Science Department
University of North Carolina

ABSTRACT

Presented is the design of a flexible expandable multiprocessor system for video graphics and image processing. The design involves a central controller which broadcasts data to a variable number of independently executing processing units, each of which in turn controls a variable number of memory units among which the video (frame buffer) image is distributed. A simple linear interconnection of processing and memory units allows straightforward sanity check detection of faulty modules and a graceful degradation of performance and/or image resolution whenever faulty processing or memory guarantees both an even work load distribution, as well as maintenance of image coherence for each processing element. Execution, speed and image resolution can be independently altered (at any time) by varying the number of processing and memory units. Sample applications of the system, for rapid line drawing and "electronic scene generation" (visible surface algorithms), are described. Variations of the design for low-cost and for powerful, real-time configurations are outlined.

1.0 INTRODUCTION*

A long-standing goal of researchers in computer graphics systems has been the development of real-time three-dimensional modeling systems. These systems, which produce a realistic image of a simulated three-dimensional environment, have a wide variety of potential uses from simulators for pilot training to interactive design of houses and automobiles. The most sophisticated of these systems produce, in real-time, images on color video displays (TV's) of startling reality. The only limitation to widespread use of these systems has been their prohibitive costs (\$500,000 and up). Thus, virtually the only uses today are those for which there is no real alternative, e.g., simulating maneuvers in gravity-free space or training simulators for pilots of large (and expensive) airplanes. If such modeling systems could be provided at significantly lower costs, it is safe to presume that their use would become dramatically more widespread.

A short examination of the computational expense of the problem suffices to justify the complexity and expense of current systems which solve it. A video image to a digital system normally consists of a matrix of picture elements ("pixels") of between 480 and 512 rows (scan lines) with from 512 to 640 pixels in each scan line. (Until recently this size was limited by the resolution of video monitors. Within the past two years, monitors with 900 to 1000 scan line capacity have become available; the factor of four increase in number of pixels per image only exacerbates the computational problem.) The image is then simply a set of some 300,000 pixels, each of which (for a color image) contains three independent components - red, green, blue - each usually to 8-bits of resolution. The

*This work was partially supported by NSF Grant MCS-77-03905, and by Naval Electronics Systems Command Contract N00039-78-C-0431, (through Research Triangle Institute Grant 43U1667). Much of this work was performed in collaboration with Brian W. Johnson of the University of Texas at Dallas while the author was a member of the faculty of that university.

entire problem at hand is simply calculating these 900,000 values each time the image is scanned out onto the video screen, usually 30 times per second.

The proper value at each pixel is a function of the data base (the simulated environment), the viewing position and orientation of the simulated viewer, and the location(s) of the light source(s) in the simulated environment. The environment is most often described as a set of objects in the environment (Euclidian three-space) coordinate system. Each object is usually described by a set of planar tiles ("polygons") which form its various surfaces, (Figure 1.1, from Sutherland, Sproull, and Schumacker (1974), shows the boundaries of a set of polygons defining the surface of a 3-D object).

Other methods of object description are sometimes used, e.g., as collections of geometric solids (MAGI (1968) or curved surfaces, Catmull (1975), Blinn and Newell 1976)). Since the particular object definition method does not significantly affect the system architecture, we shall assume hereon that the common planar polygon descriptions are used. In order to compute the red, green, and blue values for a particular pixel, the system has to determine:

- a) which, if any, polygons map onto this pixel's area,
- b) which one from this set is closest to the viewer (and thus, is the one visible obscuring all the other polygons, and
- c) the details about the precise part of this closest polygon which maps onto the pixel, its assigned color, its angle and distance from the light source(s), and its angle and distance to the viewer.

When programmed on a conventional general purpose computer, computing such a simulated image may well take several minutes, and easily longer; so developing a system to do it in 1/30 second is a non-trivial task. (The appendix gives a short synopsis of the various algorithms and approaches considered which lead to the development of the design presented in this paper.)

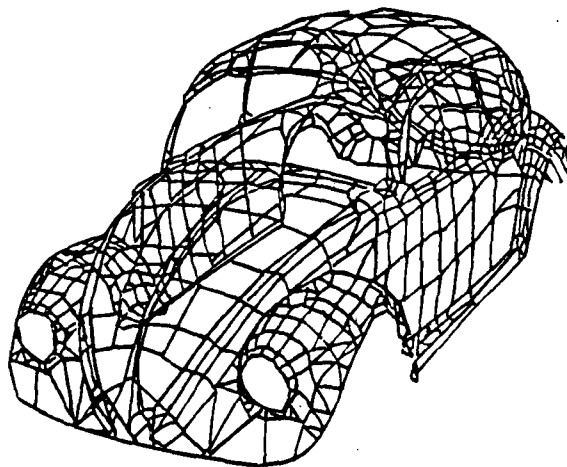


Figure 1.1 Set of Planar Tiles ("Polygons")

To understand our solution, let us first examine the overall sequence of steps which need to be performed in order to produce a visible surface image on a video display.

- a) The original polygons (in object coordinate space) are transformed into the position as seen from the simulated viewing position. (This is a sequence of rotations and translations.)
- b) The parts of the environment data base which are not in the field of view are discarded from further consideration by clipping all polygons against the boundaries of the field of view.
- c) Perspective transformation is applied to foreshorten the appropriate environmental parts as a function of distance.

It is at this point that a visible surface algorithm is invoked.

Since steps a), b), and c) can be achieved in real-time by current affordable line-drawing systems (e.g., Evans and Sutherland (1976), Vector General (1978)), we will concentrate our attention on the actual visible surface computations. (Of course, these line-drawing systems are affordable precisely because they do not have to perform the laborious visibility computations for some 300,000 pixels!) Most current real-time video systems (Evans and Sutherland (1977), Shohat and Florence (1977)) use a pipeline architecture to achieve the necessary high throughput rates. (See Figure 1.2 from Shohat and Florence (1977)). Each module in the pipeline is typically a highly specialized processing unit. Thus, these designs do not easily lend themselves to substantial upgrading (to achieve higher capacity) or downgrading (to achieve lower cost).

Our own design capitalizes on the newly plentiful resource of inexpensive LSI circuitry. Thus, each allowing a significant but bounded increase in both memory and processing requirements in return for architectural flexibility. Specifically, our solution is tailored - although not restricted - to what may be the simplest visible surface algorithm, the so-called "Z buffer" algorithm, one so simple that it seems never to have appeared in print in its own right. Sutherland, Sproull, and Schumacker (1974) mention it in passing (p. 51), saying "that if a large memory is

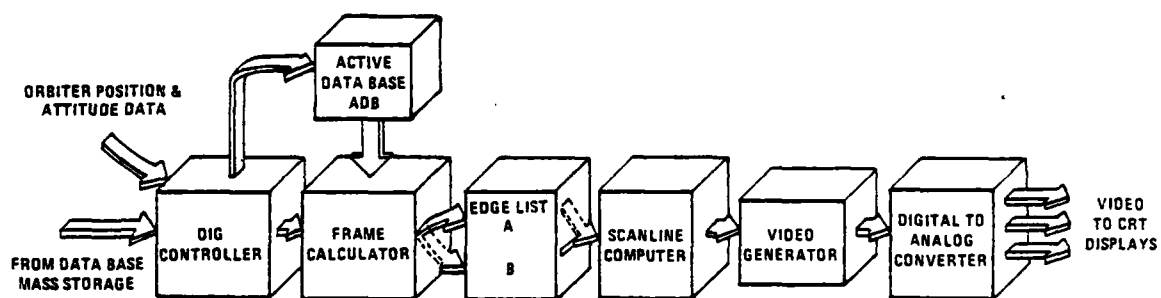
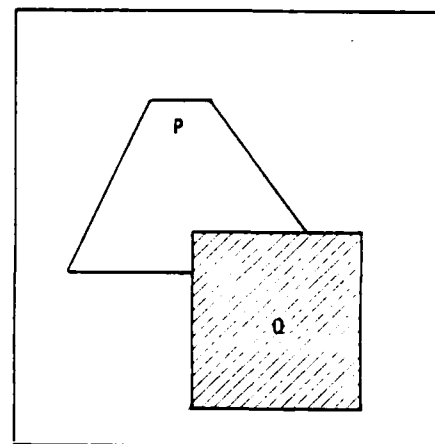
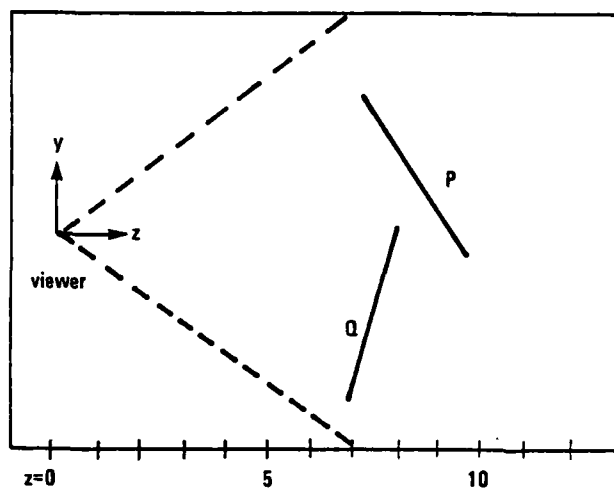


Figure 1.2 Pipeline Architecture

available . . . This method results in a computing cost which depends only on the depth number (D_c) and not otherwise on the environment complexity". (D_c is the number of front-facing polygons 'pierced, on the average, by an arbitrary ray from the viewpoint".) Catmull (1975) used the method as part of a more sophisticated algorithm for visible display of curved surfaces. The basic algorithm utilizes two large buffers each containing an entry for each pixel on the screen, an "image" buffer which contains the (RGB) intensities at each pixel, and a "Z buffer" which contains at each pixel the distance of the closest object encountered there so far (Figure 1.3). The polygons are processed sequentially, in any order. Each polygon's processing starts with determining the pixels upon which the polygon "falls" in the image. For each such pixel the distance of the polygon from the simulated viewer is computed. (This is the "Z" value.) This value is compared with the entry in the "Z buffer" for this pixel. If this new value is smaller than the current entry, then this new polygon is closer to the viewer at this pixel than the closest previously encountered polygon and so this new polygon would not be visible at this pixel. Thus, in this case the new "Z" value is put into the "Z buffer" and this new polygon's (RGB) intensity value is computed and inserted into the image buffer. If, on the other hand, the new "Z" value is greater than the value currently in the "Z buffer" at this pixel, then this polygon is farther than the closest polygon, and processing is terminated for this pixel for this polygon without any changes to the buffers. Processing continues with the next pixel into which the current polygon "falls".

This simple algorithm is seldom used, principally for two reasons: 1) few current systems have sufficient memory for two such large buffers, and 2) every pixel of every polygon needs to be computed. To understand the potential severity of this second reason, let us recall that traditionally designers of the visible surface algorithms (e.g., Watkins (1970)) have attempted to gain efficiency by avoiding, whenever possible, consideration of all but the (single) nearest polygons potentially visible on a particular scan line. These can be considered together as a set. Determining the "Z" ordering on this set at just a few key points along the scan line is sufficient to determine the sequence of visible polygon segments along the



Image

			7	7			
		8	8	8	8		
		9	9	9	9		
10	10	10	8	8	8	8	
			8	8	8	8	
			7	7	7	7	
			7	7	7	7	

Z Buffer

			7	7			
		7	7	7	7		
		7	7	7	7		
7	7	7	2	2	2	2	
			2	2	2	2	
			2	2	2	2	
			2	2	2	2	

Image Buffer
[Im(P)=7 Im(Q)=2]

Figure 1.3 Buffers

entire line (Figure 1.4). At intermediate points all the obstructed polygons are simply ignored. A "Z buffer" algorithm, since it handles each polygon separately, computes every affected pixel for each polygon, a procedure which certainly seems to be wasteful and inefficient. However, a closer examination of the situation reveals that for multiprocessor systems the procedure may in fact be very attractive. Sutherland, Sproull, and Schumacker (1974) estimate that the average number of polygons "falling on" a pixel is only 3; that is, many images contain large areas of sky, water, ceiling, floors - areas in which there are not too many polygons stacked one behind the other. This implies that the (in)efficiency of the "Z buffer" algorithm is constant; at worst it is some constant multiple (e.g., 3) of the most efficient possible algorithm, one which can determine with negligible cost the visible polygon at each pixel. Since LSI technology is rapidly diminishing the cost of simple arithmetic processing units, a factor of 3 is no longer burdensome.

2.0 SYSTEM DESCRIPTION

The fundamental system organization is as illustrated in Figure 2.1. Figure 2.2 shows in somewhat greater detail the organization of the image buffer, which is accessed by both the processor and the video scan generator. Figure 2.3 illustrates the simple time division multiplexing between the processor and the video scan generator. We note here that the current pixel's data remains on the video scan generator bus even during the period which is assigned to the processor.

If we consider using only commonly available inexpensive LSI RAM's, then the requirement of the scan generator (needing to cycle through the entire image in approximately 30 milliseconds) will limit the usefulness of this simple design to very coarse images. To increase the bandwidth we simply insert additional memory units onto the system bus. Figure 2.4 illustrates the organization of this enhancement and Figure 2.5 shows the timing cycles. It is important to note that the actual bus to the scan generator does not increase in size or speed. All memory units are read in parallel during the scan generator access times. During the following

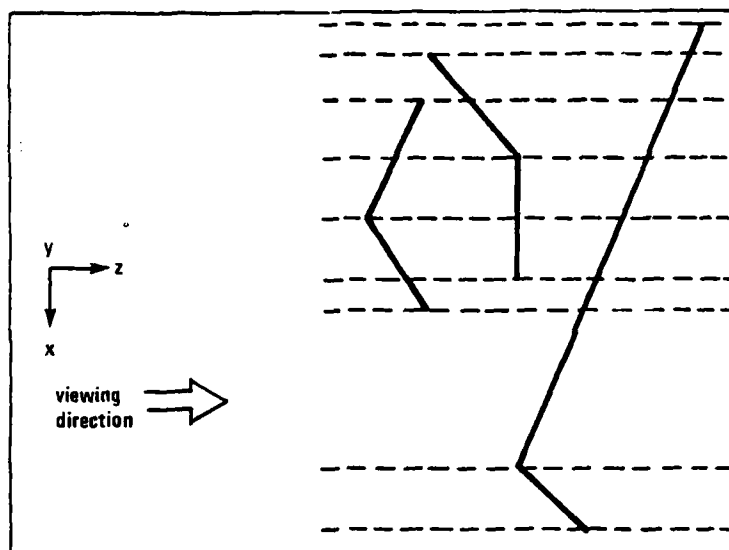


Figure 1.4 Key Point of Scan Line

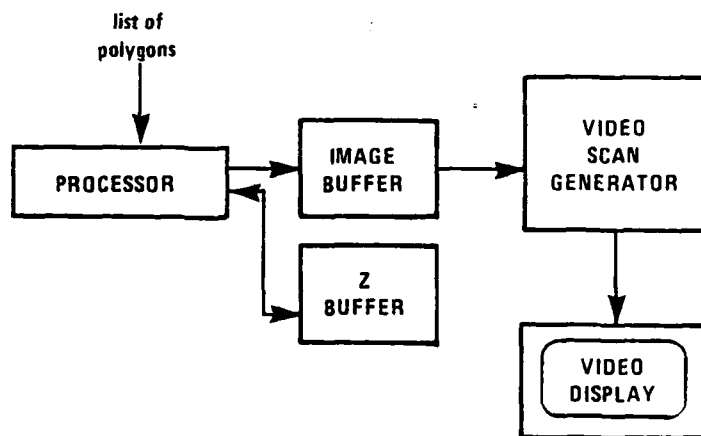


Figure 2.1 Fundamental System Organization

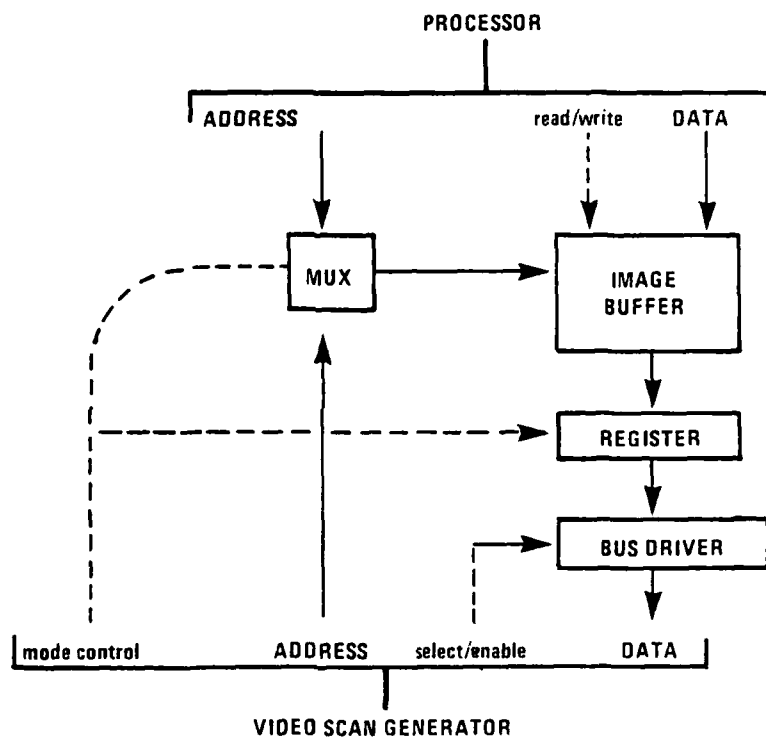


Figure 2.2 Organization of the Image Buffer

ACCESS MODE

VIDEO SCAN
GENERATOR

PROCESSOR

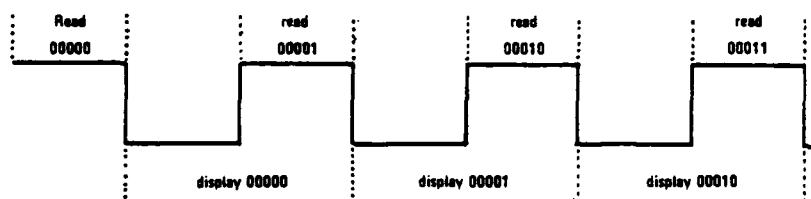


Figure 2.3 Memory Unit Timing

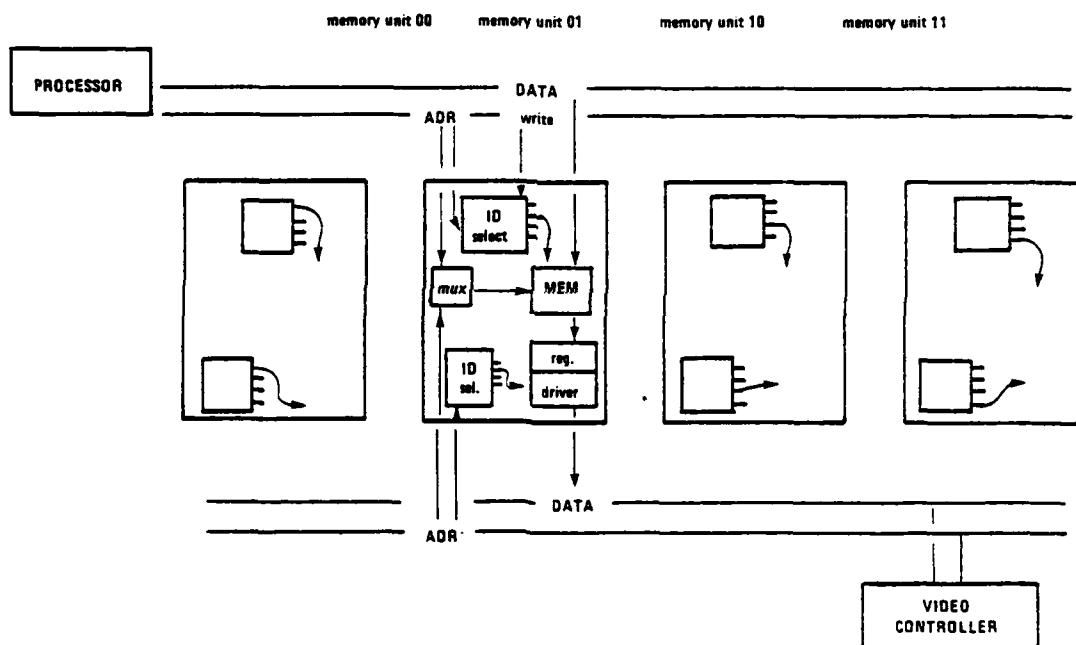
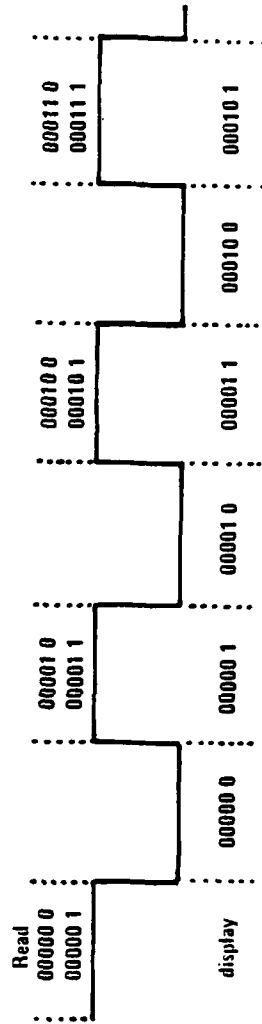


Figure 2.4 Organization of Additional Memory Units

ACCESS MODE

VIDEO SCAN
GENERATOR

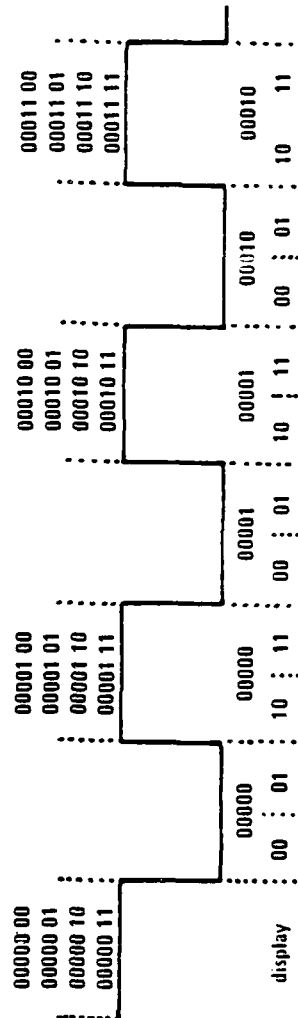
PROCESSOR



TIMING - 2 MEMORY UNITS

VIDEO SCAN
GENERATOR

PROCESSOR



TIMING - 4 MEMORY UNITS

Figure 2.5 Timing Cycles of Additional Memory Units

complete timing cycle, the various results are put onto the video bus by enabling, in sequence, the bus drivers of the various memory units. This enabling is directly controlled by the least significant bits of the video scan generator's X address. In this fashion the number of memory units need not be known to the scan generator. If there are fewer units, some of the least significant address bits are ignored and thus, consecutive locations on the video screen will be accessed from the same image memory unit's output register. The result will be a coarser image (128 x 128, say instead of 512 x 512) than the scan generator is capable of producing. (It will be seen later that a somewhat different resolution-independence scheme for the processor side of the memories will free the entire system, both hardware and software, from reliance on a fixed resolution.) The proper ID selection in each memory unit (as seen in Figure 2.3) is a function of both the unit's ID number and the total number of memory units currently in the system. Although such selection settings are normally set manually through jumpers or DIP switches, we prefer for them to be set automatically. This is done through the following mechanism. In addition to the processor bus and video scan generator bus, the system includes a set of lines for ID numbers and the "total-units" number.

As illustrated in Figure 2.6, the ID lines consist of a set of lines sufficient to represent the largest possible number of memory units in a system. (For example, for a 1024 maximum memory unit system this number would be 10.)

In this fashion the set of lines are started at 0 on one side, each board has an increment circuit on it, thus the number on the backplane ID lines is incremented by one each time it passes through a memory unit board (Figure 2.7). A similar set of lines is used to return the ID signal value from the end of the system. (This number is simply the total number of memory units in the system at the present time.) With this technique boards can be inserted into or extracted from any position at any time without the necessity of any hardware (or software!) modification.

We also note at this point that neither the video scan generator nor the image memories rely on any mechanism for altering the contents of the image memories. Thus, we can distribute responsibility for computing the image memories contents to a number of different processors.

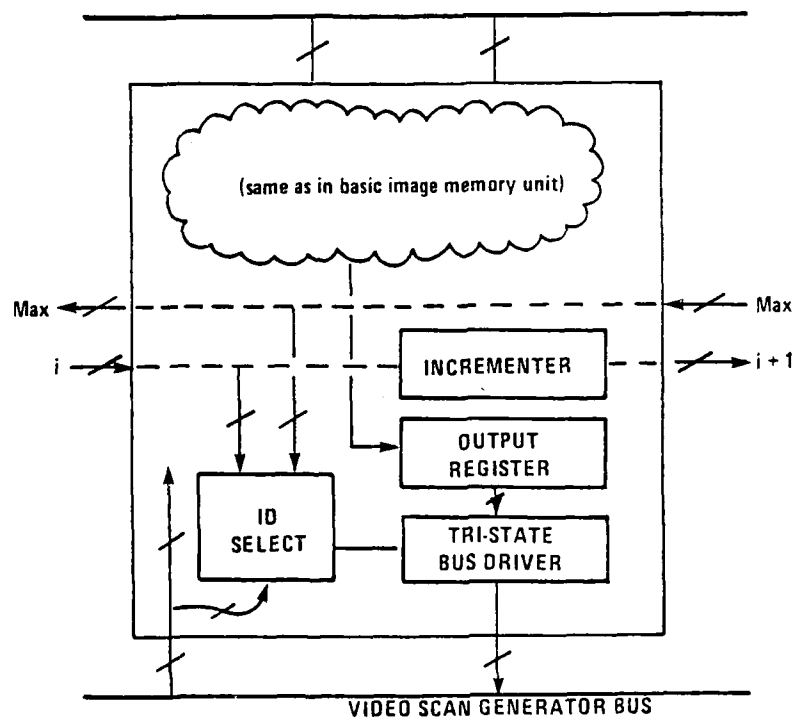


Figure 2.6 Memory Address Structure

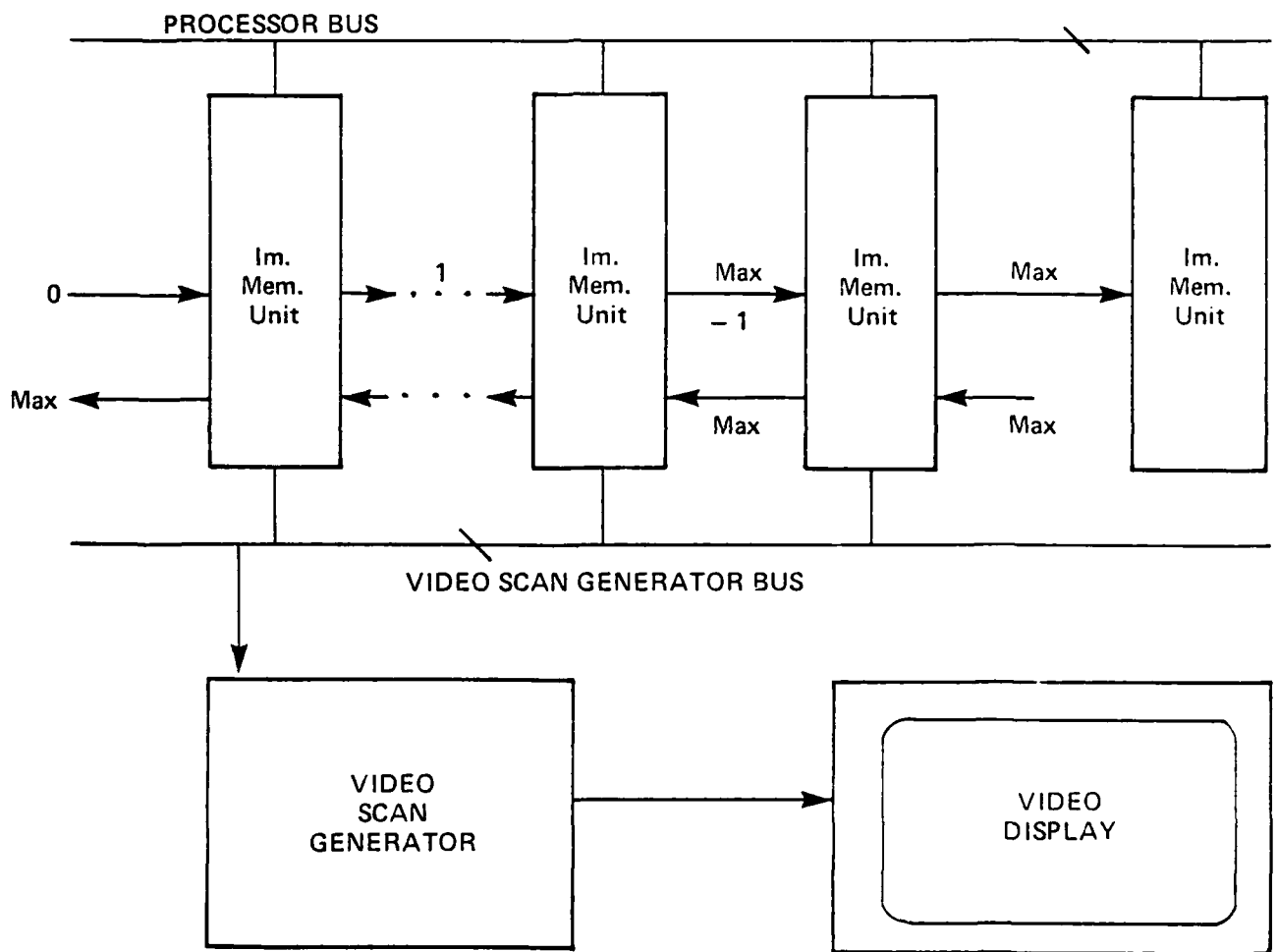


Figure 2.7 Unit Interconnection Diagram

Figure 2.8 illustrates a modified organization which achieves this increased capability. Virtually the only addition has been the introduction of a central broadcast controller (CBC) which "announces" the description of each new polygon to all the processing elements (PE's). The system is designed to operate as follows:

- a) Immediately upon power-on, the CBC broadcasts the (possibly new) software to all the processing elements. (All PE's execute the same program, but each has a separate copy of it and each may be executing different parts of it at any instant.)
- b) The CBC instructs the PE's to survey the memory units under their control. This consists simply of each PE attempting to read and write a single word into each possible memory unit under its control. (Each knows (from the ID lines), 1) the total number of units in the system at this time, and 2) the first memory unit that is under its control; it simply needs to find the upper limit of its domain.)
- c) The "Z" and image buffers are initialized by each PE.
- d) The actual processing proceeds now with the CBC broadcasting description of one or more polygons to be processed. Since each PE knows which MU's are under its responsibility and how many MU's are in the system, it can easily compute the location of each of its pixels on the screen. For each polygon it does the appropriate "Z buffer" algorithm computations (as outlined before) for all its pixels affected by this current polygon. When done, each PE signals to the CBC. When all the PE's are done, the CBC broadcasts the next polygon (or set of polygons). The procedure continues until the complete set of polygons in the scene are exhausted.

By having the MU's and the PE's implemented on the same size PC cards and utilizing the same connectors, all the PE bus lines can be implemented on a single set of backplane lines. A PE simply ignores any such signals coming in from its left, and generates its own signals on the lines to its

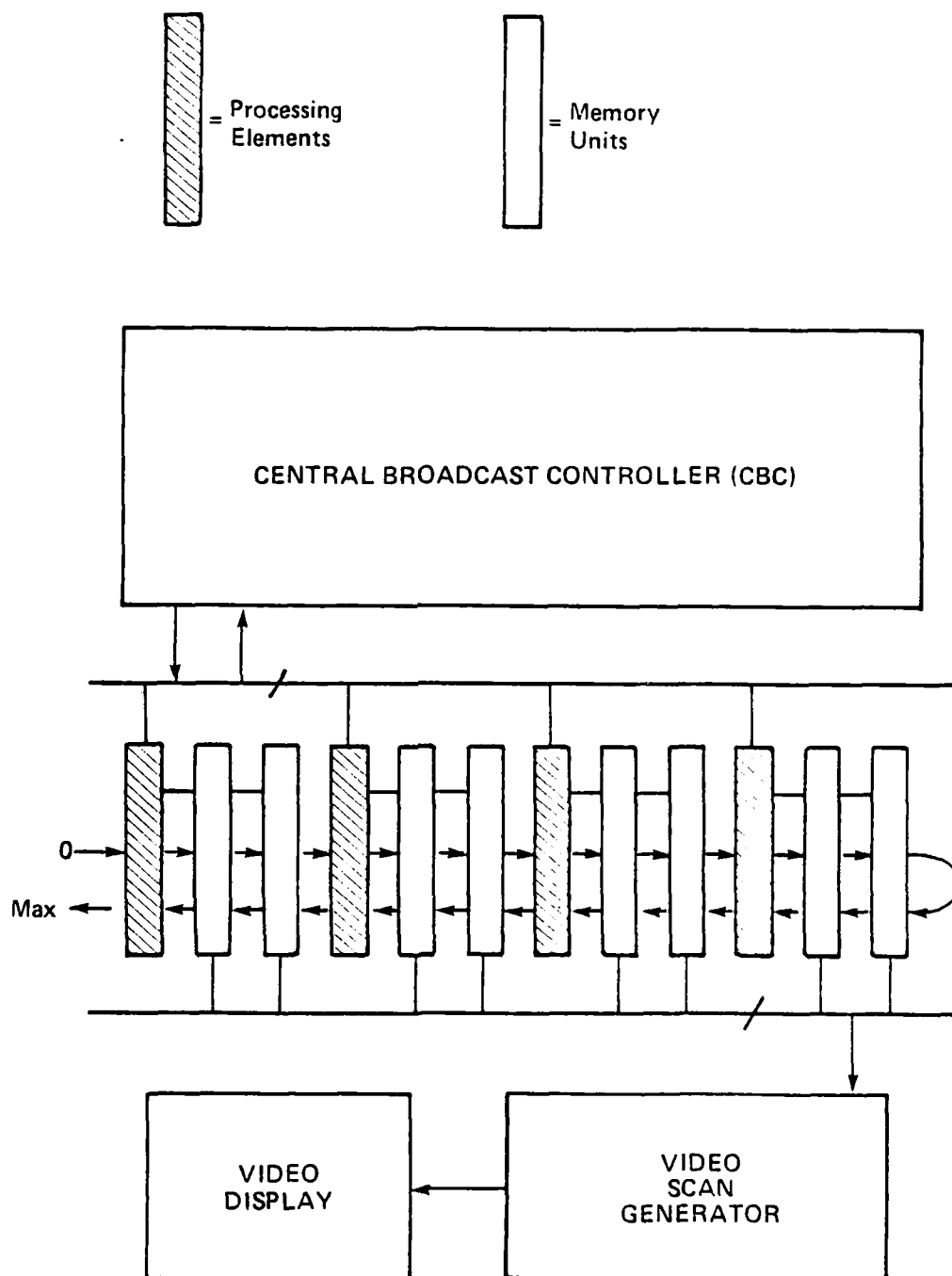


Figure 2.8 Modified Organization

right. (ME's simply pass these signals through.) Thus, a PE simply controls all the MU's between it and the next PE on its right. Configurations can be altered by simply reshuffling the boards.

Figure 2.9 illustrates that regular interlacing is possible for both the MU's and the PE's. This is particularly important for efficient processing, for two reasons: 1) it guarantees that for any polygon (of size greater than a single pixel and for systems with greater than two processors) the pixels on which it lies will be located in the domains of a number of different PE's so that the work load will always be distributed, and 2) the regular pattern of affected pixels in any one MU allows rapid incremental computations for "Z", and eventually for RGB. (Recall that all these polygons are planar; so the amount of change per each pixel step will be constant.) Also, the same regular pattern occurs in each affected MU; for example, if adjacent pixels in a particular MU are 2 units apart in X and 4 units in Y, then they will be that way for every affected memory unit. This allows the CBC to compute the appropriate incremental change values during the time the PE's are processing the previous polygon. The CBC can then broadcast these values directly, thereby avoiding a computation step in each PE.

Figure 2.10 shows how a particular configuration can be modified to increase or decrease image resolution or processing speed. (The variations in processor memory assignments from those of Figure 2.9 reflect the computations performed by the memory ID select modules illustrated in Figure 2.6.) Figure 2.11 illustrates the physical organization corresponding to the various resolution/speed configurations of Figure 2.10.

Let us consider some of the capabilities of this kind of organization. It allows virtually limitless flexibility in tradeoff between power and economy. On the one extreme there can be systems with only one PE and one MU. Of course, such a system would exhibit a very coarse image, but it may be suitable for simple video games, for instance. On the other extreme one can configure a system with high resolution and very high throughput. The number of pixels per PE can be reduced all the way down to one (although this seems impractical), thereby allowing a polygon to be processed within microseconds. Such high resolution and high powered systems would be

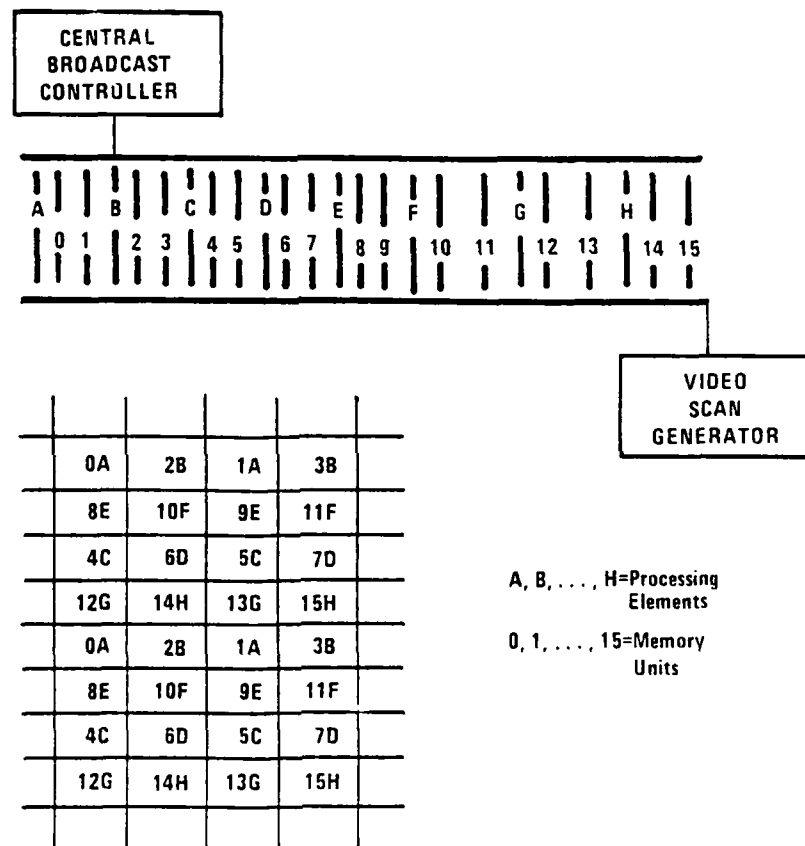


Figure 2.9 Regular Interfacing for MU and PE

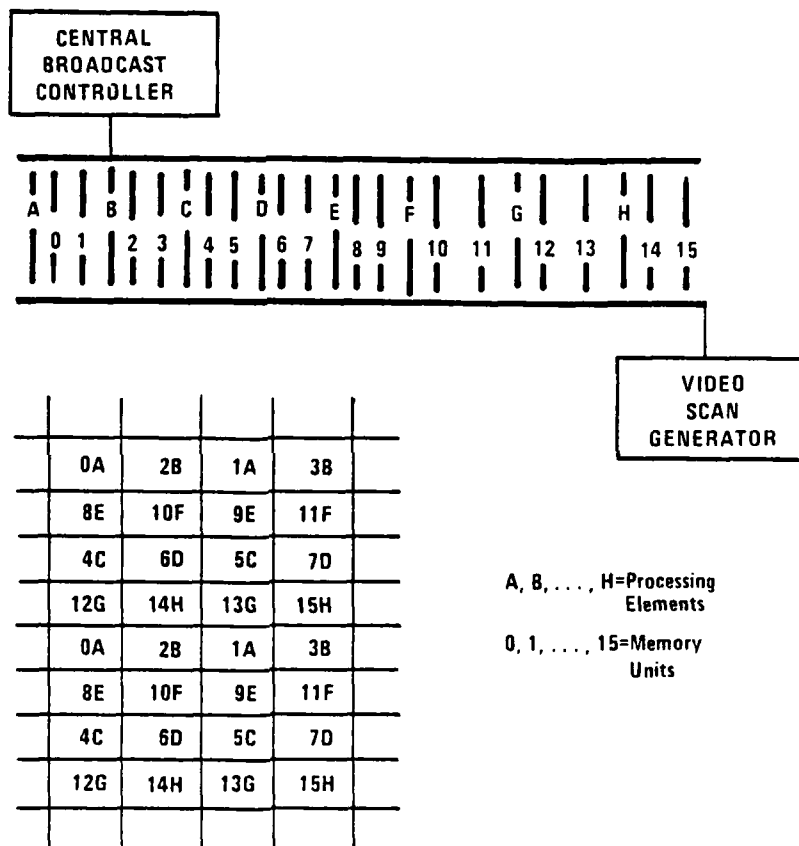


Figure 2.10 Modification to Increase or Decrease Image Resolution of Processing Speed

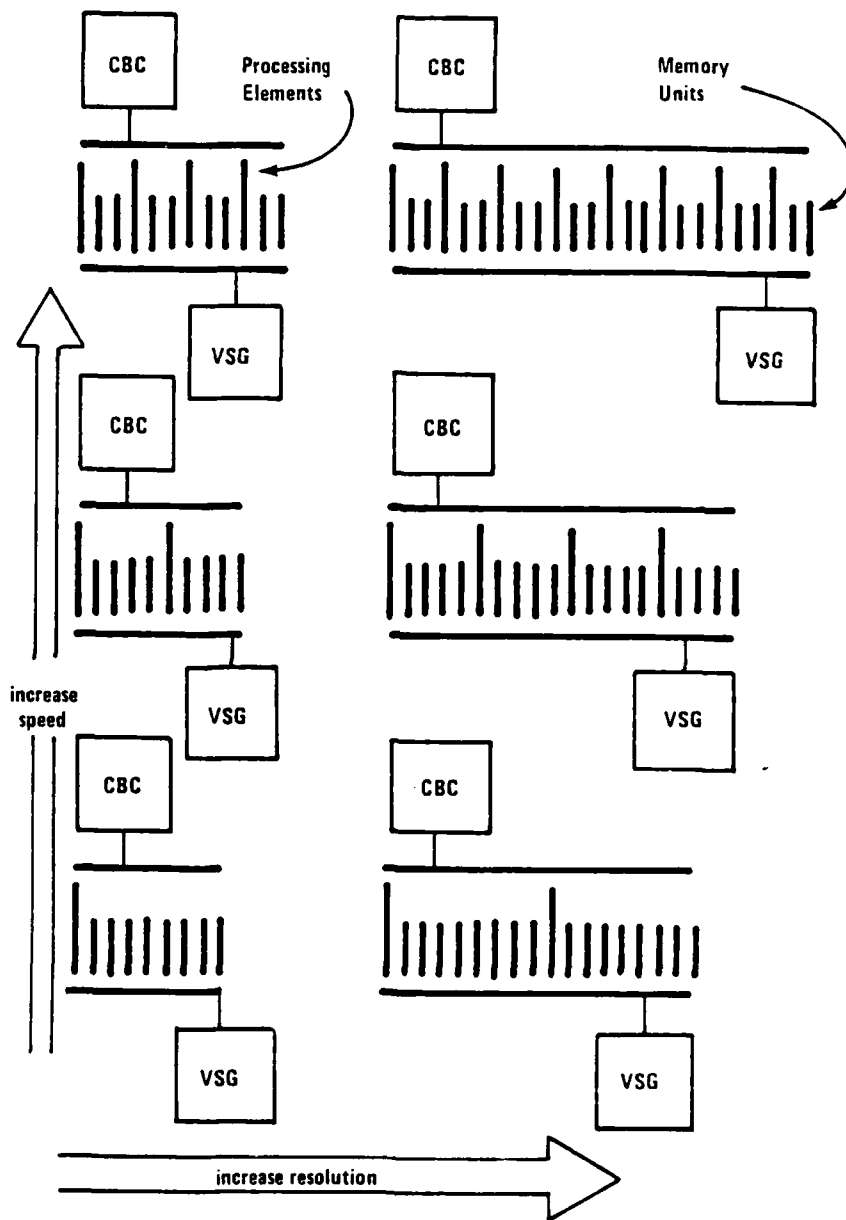


Figure 2.11 Illustration of Physical Organization Corresponding to Various Resolution/Speed Configurations of Figure 2.10

appropriate, for instance, for real-time pilot-training simulators. The only difference, however, between these two extreme configurations would be the number of PE boards and the number of MU boards. The software in the PE's of both systems would be identical. The CBC's would be identical. (The polygons are broadcast in highest resolution units; low-resolution configurations simply ignore some of the least significant bits.) The video scan generators could also be identical. (They also run high-resolution counters; small systems again simply ignore some least significant bits.) It is reasonable to speculate that a large computing facility may have a number of machines, each with a different number of MU and PE boards, many small configurations for program development, a few large ones for real-time simulation, and some high resolution but slow ones for non-timecritical applications. For special occasions, larger configurations could easily be constructed by simply consolidating several small configurations. Also, faulty boards could simply be removed from a system.

These systems should also degrade gracefully. Some current real-time systems encounter difficulty due to computations being done "on the fly" as the video beam scans the image. Thus, these systems avoid using an image buffer between the processing and scanning-out modules. If a certain spot in the image is particularly complex, however, the scan either has to wait, or it "paints" incorrect data. The design presented here would not exhibit such behavior. The system would simply take slightly longer to compute the new image. If the memories were double buffered, the switch between the old image and the new one would be made slightly after the start of the second scan of the old image, or if the situation were really complex, the switch would be made after two or more complete scans of the (old) image.

3.0 BUILT-IN-TESTING AND FAULT TOLERANCE

Since the overall utility of these kinds of systems is to a large extent a function of availability and "up" time, it is essential to consider the possibility of faulty components and the consequences of such to the viability of the overall system. The goal, of course then, is to keep the system (correctly) operational whenever possible. This involves the design

of built-in-test facilities to detect and isolate and if at all possible, to establish recovery procedures. Since the system is primarily composed of but two distinct modules, PE's and MU's, we will discuss each of them separately.

One approach under consideration is to adopt standard double or triple redundancy for either error detection or error correction, respectively (FT reference here). A doubly-redundant approach, with a comparison test module, would be particularly easy to implement. As described in the preceding section, the design currently adopts the convention that each PE controls the MU's immediately to its right up to the position of the next PE and ignores all the signals going in on its left from the preceding PE (see again Figures 2.8 and 2.9). We can easily modify this convention by inserting additional PE's, for instance, as illustrated in Figure 2.10. In such a configuration the new PE's, which may physically be identical to the other PE's are indicated as TPE - 0 through TPE - n. The processing is modified as follows: The initialization procedure (during which time each PE determines the MU's under its control) is expanded to include a second phase during which time each RPE determines the MU's under its "control", which are not to their left. (During this procedure the TPE's are idle, simply passing all signals through.) During normal processing the standard PE's operate as before, the RPE's operate as if they were standard PE's with the sole exception that they generate their signals to the left and not to the right. Each TPE then, simply compares the signals coming to it from its left and right, an error condition is indicated when they are not identical.

Since we may not normally expect a very high fault rate for PE, we may be satisfied at a less expensive solution, one which utilizes each PE as an RPE and to function as an RPE and a TDE for its neighbor. This approach would operate as follows. The ends of the backplane bus would be physically interconnected, giving the system a circularly linked interconnection of PE's. The initialization procedure is extended to include the communication of the just determined memory assignments of each PE to its right neighbor. This can easily be achieved under CBC supervision since the PE controller lines on each PE's left are received (although ignored most often).

Once the memory assignment of a PE is known to its right neighbor, this neighbor can perform precisely the same algorithm calculations as its left neighbor and match them against the signals it receives from its left. Since a PE can either check on its neighbor or perform its own processing, normal generation would not be divided into two phases: a) even numbered PE's operating as usual and odd numbered PE's checking and b) odd PE's operating as usual and even PE's checking.

Of course, this is basically double redundancy without the test PE's, and would achieve the same performance. (All processing would have to show a BIT to allow the checking PE's to match each of their results with the left neighbor's.) The flexibility gained is that the half-running, half-checking operation mode does not need to be invoked all the time. If the system is judged sufficiently reliable and/or the system's users can tolerate some short-term faults, e.g., images overlayed with some erroneous "garbage", then normal processing may be utilized, say 90% of the time and "checking and running" only the other 10%. Since this ratio could be dynamically adjusted under software CBC direction, the system's speed vs. reliability can be set appropriately for each specific task and/or based on recent system performance.

Memory fault-tolerance can be similarly "tuned" by adding additional MU's and having the scan generator enhanced to be able to compare the two values it receives for certain pixels. In such a version, the scan generator would be physically located in the middle of the overall system.

In such a configuration it would likely be advantageous to avoid completely duplicating the original system. One solution would be to have the "checking" system's video image spacing arranged to duplicate some pixel locations in each of the standard system's MU's. In this way, the amount of memory redundancy could be controlled by varying the ratio of the number of MU's in the original and "checking" system. The enhanced VSG would now have to know which pixels were in fact duplicated and would have to know when to check, since the speed of the two independent systems would likely not be identical. With either PE or MU components, once a faulty unit is identified it can either be replaced on site or simply removed

from the system. Of course, with one fewer PE or MU, the system will either execute somewhat slower (in the case of faulty PE, was removed).

Design of built-in-testing and fault tolerance of the remaining two single modules the Central Broadcast Controller (CBC) and the Video Sync Generator (VSG) are still pending. Two present advantages are 1) that these modules comprise but two out of perhaps dozens in a typical configuration, and 2) that the overall system design allow these to be implemented via the same physical board site and bus connections as the other modules so that on-site replacement would be virtually as simple as for the other system modules.

4.0 OTHER APPLICATIONS

It is easy to see at this point that the system is not restricted to simply executing a "Z buffer" visible surface algorithm. Software could be loaded into the PE's, for instance, to perform digital vector generation and rapidly create line drawings on the video screen. In this case, the CBC would simply broadcast endpoint information, each of the PE's would determine the pixels under its control which are affected by the new line segment; it would then set each of these pixels appropriately.

5.0 IMPLEMENTATION

We are currently in the process of implementing various aspects of the above design. We have prototyped simple versions of each module and plan to have a small, but complete prototype system in the near future.

REFERENCES

1. Appel, A. (1967), "The Notion of Quantitative Invisibility in the Machine Rendering of Solids", Proc. ACM Annual Conference, pp. 387-393.
2. Blinn, J. F. and M. E. Newell (1976), "Texture and Reflection In Computer Generated Images", Comm. ACM 19 (10), pp. 542-547.
3. Bouknight, W. J. (1969), "An Improved Procedure for Generation of Half-Tone Computer Graphics Representations", University of Illinois, Coordinated Science Laboratory, R-432.
4. Catmull, E. A. (1975), "Computer Display of Curved Surfaces", Proc. Conference on Computer Graphics: Pattern Recognition and Data Structures, (IEEE Cat. No. 75CH0981-1C), pp. 11-17.
5. Despain, A. M. and D. A. Patterson (1978), "X-Tree: A Tree Structured Multiprocessor Computer Architecture", Proc. Fifth Annual Symposium on Computer Architecture, pp. 144-150.
6. Evans and Sutherland Computer Corporation (1976), "Picture System 2", Salt Lake City, Utah.
7. Evans and Sutherland Computer Corporation (1977), "Improved Scene Generation Capability", Final Report, NASA Contract No. NAS 9-14010.
8. Hirschberg, D. S. (1978), "Fast Parallel Sorting Algorithms", Comm. ACM 21 (8), pp. 657-661.
9. MAGI (1978), Mathematical Applications Group, Inc. Elmsford, N.Y., Promotional Literature.
10. Newell, M. A., R. G. Newell, and T. L. Saucha (1972), "A New Approach to the Shaded Picture Problem", Proc. ACM Annual Conference.
11. Roberts, L. G. (1963), "Machine Perception of Three-Dimensional Solids", MIT Lincoln Laboratory, TR 315. Also in Optical and Electro-Optical Information Processing, Tripper, et.al., eds. MIT Press, 159.
12. Rougelot, R. S. and R. Schumacker (1969) G. E. Real-Time Display, NASA Report NAS 9-3916, General Electric Co., Syracuse, N.Y.
13. Shohat, M. and J. Florence (1977), "Application of Digital Image Generation to the Shuttle Mission Simulation", Proc. 1977 Summer Computer Simulation Conference.
14. Schumacker, R. A., B. Brand, M. Guilliland, and W. Sharp (1969), "Study for Applying Computer-Generated Images to Visual Simulation", U.S. Air Force Human Resources Laboratory, AFHRL-TR-69-14.

REFERENCES (Continued)

15. Sutherland, I. E., R. F. Sproull, and R. A. Schumacker (1974), "A Characteristic of Ten Hidden-Surface Algorithms", ACM Computing Surveys, 6 (1), pp. 1-55.
16. Vector General, Inc. (1978), System 3300, Woodland Hills, CA.
17. Warnock, J. E. (1969), "A Hidden Surface Algorithm for Computer-Generated Halftone Pictures", Computer Science Department, The University of Utah, TR 4-15.
18. Watkins, G. S. (1970), "A Real-Time Visible Surface Algorithm", Computer Science Department, The University of Utah: UTECH-CSC-70-101.
19. Wieler, K. and P. Atherton (1977), "Hidden Surface Removal Using Polygon Area Sorting", Proc. Fourth Annual ACM SIGGRAPH Conference on Computer Graphics and Interactive Techniques, pp. 214-222.

APPENDIX

A short survey of the applicability of various visible surface algorithms for distributed processing will aid in understanding the approach we have developed for our own design.

Sutherland, Sproull, and Schumacker (1974) classify the various visible surface algorithms into object space, image space, and list priority algorithms. Object space algorithms (e.g., Roberts (1963), Appel (1967)) process the environment's object parts sequentially and determine, for each such part, whether or not it is visible. Image space algorithms, (e.g., Bouknight (1969), Watkins (1970)), on the other hand, take each part of the image sequentially and determine for each such image area; eventually a single pixel which object parts is visible there. List priority algorithms, e.g., Schumacker, et. al. (1969). Newell, et. al. (1972) determine some ordering on the list of polygons in the environment, either from farthest to closest to the viewer or some other arrangement based on geometric relations between the polygons. With such an approach the visible polygon at each pixel is simply the highest priority polygon which maps onto it.

Let us consider the suitability of these various approaches for distributed execution. An obvious approach for distributing work load of an object space algorithms would be to divide the various objective parts between the available processors in the system. This approach would encounter difficulty in at least two places: in order to determine the visibility of any object part, possibly all the other objects would have to be examined. Thus, each processor would need to have constantly available the entire set of possibly visible polygons. In addition to this, the results of all the visibility calculations need to be put onto the screen. The two alternatives for this part are, a) to have a real-time scan generator which calculates the intensity values as the video beam is scanning the display screen, or b) to have an image memory buffer ("frame buffer") in which the image pixels are put as they are determined and have the image scanned out from this buffer (See Figure IV-A.1).

The first alternative would certainly be difficult in this case, since the values for the scan would be randomly distributed among the various processors, and, in general, even in a single process, the scan order of

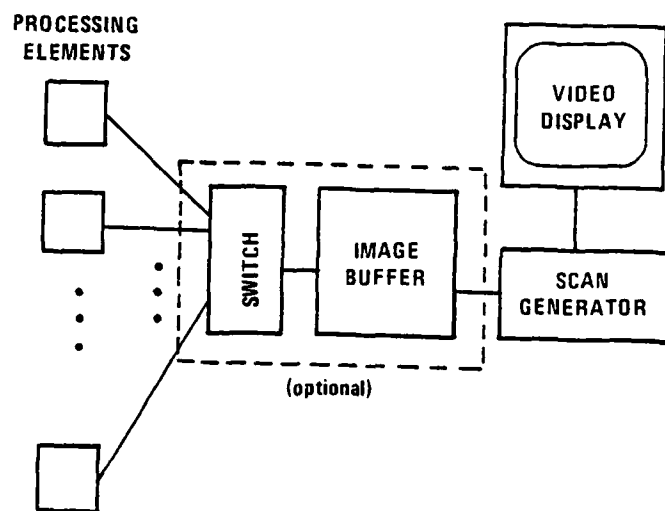


Figure IV-A.1 Real-Time Image Buffer

the various object parts would need to be scan order, not in object-space order. The alternate approach, that of putting the results from the various processors into a frame buffer from which the scan generators "read out" the image, would most likely suffer from excessive contention for the frame buffer. As the various processors all attempt to write all their information into the frame buffer, the bandwidth of a large random access buffer (assuming 700 msec cycle time and 50% time division multiplexing between the scan generator and the image-determining processors) leaves less than 12,000 total accesses for all the processors during each frame time. One may wish to partition the frame buffer into a number of smaller units in order to overcome this bandwidth limitation, but since each processor's visible object parts can be expected to be randomly distributed in the image, there will then need to be data paths between each processor and each memory (see Figure IV-A.2).

List priority algorithm may be more applicable to distributed processing. In fact, one of the earliest real-time systems (GE) is based on a list-priority algorithm. This particular type of priority is based on a geometric relationship between object polygons and, as such, needs only to be calculated once for a rigid environment and is largely independent of the simulated viewing position. To calculate this relationship, however, the system often needs expert manual intervention to modify the environment's definition. This requirement significantly detracts from the appeal of this approach. The other well-known list of priority algorithm Newell, et. al. (1972) orders the list of polygons from back to front, from the polygon farthest from the viewer to the one closest to the viewer, then "paints" the polygons into the frame buffer in this order. A polygon which obscures another one behind it would be encountered after the obscured one in the ordered list; it would thus "paint over" the more distant polygon.

The applicability of this approach to distributed processing is certainly not obvious since the major step is a rather elaborate sort involving the entire set of potentially visible polygons. Although parallel sorting methods may be useful here (Hirschberg (1978)), the situation is complicated by the lack of a single sorting key. The sort, rather, involves the "visibility priority" or the "obscuring level" of the various

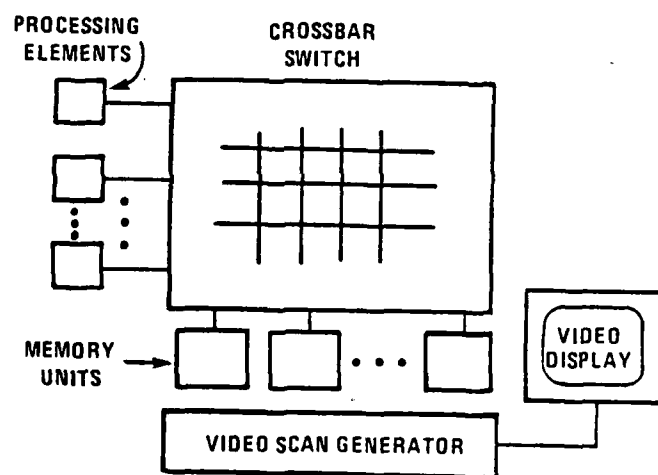


Figure IV-A.2 Frame Buffer Partitioned into Smaller Units

polygons. The required condition is that if polygon A obscures polygon B then A must not be placed before B in the "painting" list. It is simple to demonstrate that this kind of an ordering may not even exist for some sets of polygons (see Figure IV-A3.). In such cases, polygons have to be split into pieces until a strict ordering can be established. Even if such an involved sorting could be distributed over multiple processors, the basic method of determining visibility by "painting over" nearer polygons seems to imply a sequential process moving from back to front. The list could, of course, be split separately computed by a single processor with a separate image buffer. The final image would then consist of the various image buffers merged in the appropriate priority order by the scan generator (see Figure IV-A.4). The expense of a full image buffer with each processor makes this approach rather impractical.

Object space algorithms are rather more appealing for possible distributed processing. An obvious approach would be to distribute the work load among the various processors by partitioning the image between them. Scan line order algorithms, such as Watkins (1970), could be implemented in this fashion by assigning various scan lines to different processors. The complicated nature of these algorithms and their reliance on incremental processing, calculating one scan line is basically a modification of the data on the previous scan line, makes this approach difficult.

The algorithm by Warnock (1969) basically considers the set of polygons involved in a particular area of the screen. If there are too many, then it partitions the area into (usually) four regions, creating a larger number of problems to solve, but each of them simpler to solve than their common predecessor (or at the very least no more complicated). Infinite recursive subdivision is avoided by the realization that once the area is that of a single pixel, the system can simply find the closest polygon. The algorithm capitalizes on the characteristic that almost all images contain many empty and many very simple regions.

This approach seems difficult to adopt for distributed processing since the work load is a function of area, but the areas are not evenly distributed across the full image. Although some appropriately interconnected network of processors could possibly be used to solve the

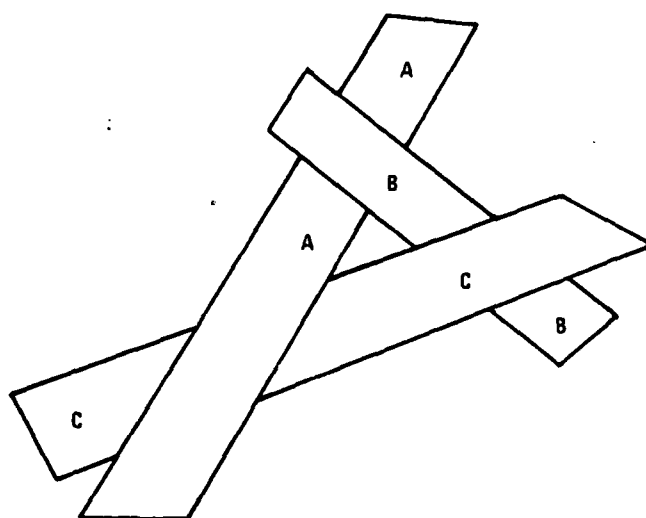


Figure IV-A.3 Obscuring Polygons

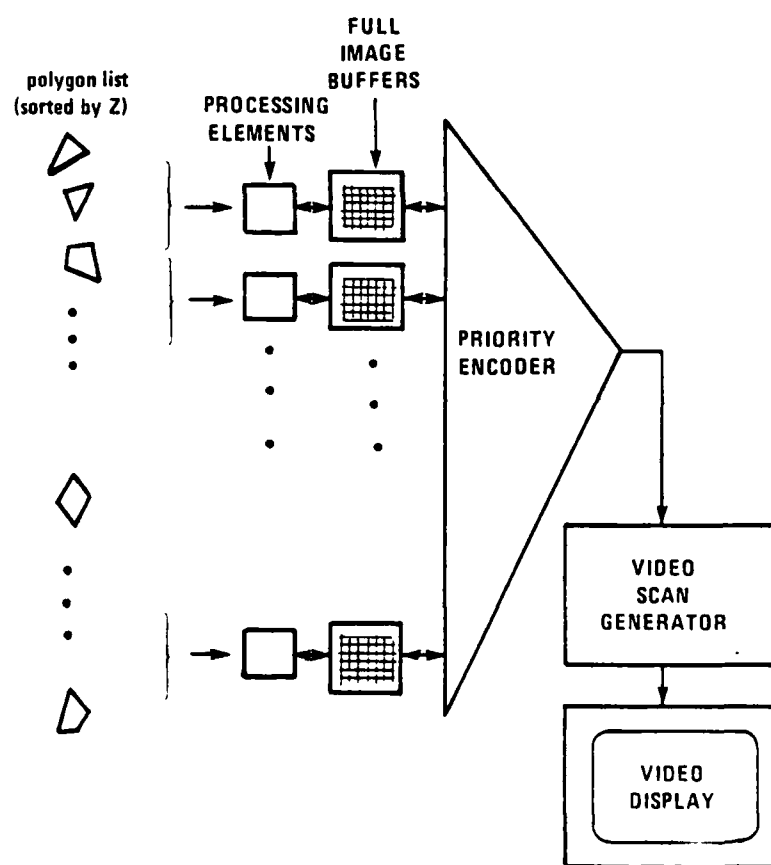


Figure IV-A.4 Priority Order of Various Image Buffers

visibility problem in this fashion (Dispain and Patterson (1977)), with one processor activating several others whenever an area is subdivided, it seems that the contention for the image buffer by the various processors would still remain as intractable as before.

A recent visible surface algorithm by Weiler and Atherton (1977) is in some ways an appealing combination of that by Warnock (1969) and that by Newell, et. al., (1972), but seems equally difficult to adapt to a distributed organization for some of the same reasons as those of its mentioned predecessors.